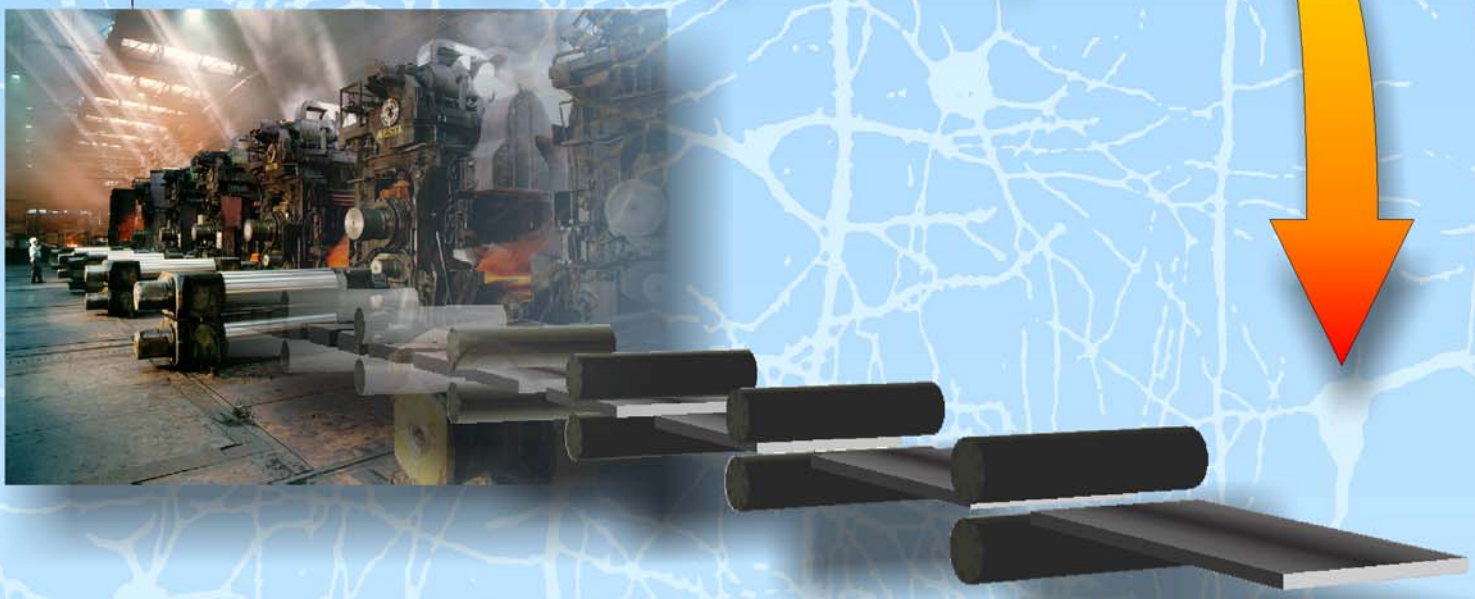

Neuronale Netze zur Berechnung Iterativer Wurzeln und Fraktionaler Iterationen

Lars Kindermann

Neuronale Netze zur Berechnung von Iterativen Wurzeln und Fraktionalen Iterationen

Von der Fakultät für Elektrotechnik und Informationstechnik
der Technischen Universität Chemnitz genehmigte

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften

Dr.-Ing.

vorgelegt von

Dipl.-Phys. Lars Kindermann
geboren am 25. März 1963 in Uelzen

Eingereicht am 4. Juli 2001

Gutachter:

Professor Dr. Peter Protzel, Chemnitz
Professor Dr. Günter Radons, Chemnitz
Professor Dr. Manfred R. Schroeder, Göttingen

Tag der Verleihung: Chemnitz, den 3. Juli 2002

Zusammenfassung

Diese Arbeit entwickelt eine Methode, Funktionalgleichungen der Art $\varphi(\varphi(x)) = f(x)$ bzw. $\varphi^n(x) = f(x)$ mit Hilfe neuronaler Netze zu lösen. Gesucht ist eine Funktion $\varphi(x)$, die mehrfach hintereinandergeschaltet genau einer gegebenen Funktion $f(x)$ entspricht. Man nennt $\varphi = f^{1/n}$ eine iterative Wurzel oder fraktionale Iteration von f. Lösungen für φ zu finden, stellt das inverse Problem der Iteration dar oder die Erweiterung der Wurzel- bzw. Potenzoperation auf die Funktionsalgebra. Geschlossene Ausdrücke für Funktionswurzeln einer gegebenen Funktion zu finden, ist in der Regel nicht möglich oder sehr schwer. Numerische Verfahren sind nicht in allgemeiner Form beschrieben oder als Software vorhanden.

Ausgehend von der Fähigkeit eines neuronalen Netzes, speziell des mehrschichtigen Perzeptrons, durch Training eine gegebene Funktion $f(x)$ zu approximieren, erlaubt eine spezielle Topologie des Netzes auch die Berechnung von fraktionalen Iterationen von f. Ein solches Netz besteht aus n identischen, hintereinandergeschalteteten Teilnetzen, die, wenn das Gesamtnetz f approximiert, jedes für sich $\varphi = f^{1/n}$ annähern. Es ist lediglich beim Training des Netzes darauf zu achten, dass die korrespondierenden Gewichte aller Teilnetze den gleichen Wert annehmen. Dazu werden mehrere Verfahren entwickelt: Lernen nur im letzten Teilnetz und Kopieren der Gewichte auf die anderen Teile, Angleichen der Teilnetze durch Kopplungsfaktoren oder Einführung eines Fehlerterms, der Unterschiede in den Teilnetzen bestraft.

Als weitere Näherungslösung wird ein iteriertes lineares Modell entwickelt, das durch ein herkömmliches neuronales Netz mit hoher Approximationsgüte für nichtlineare Zusammenhänge korrigiert wird.

Als Anwendung ist konkret die Modellierung der Bandprofilentwicklung beim Warmwalzen von Stahlblech gegeben. Einige Zentimeter dicke Stahlblöcke werden in einer Walzstraße von mehreren gleichartigen, hintereinanderliegenden Walzgerüsten zu Blechen von wenigen Millimetern Dicke gewalzt. Neben der Dicke ist das Profil - der Dickenunterschied zwischen Bandmitte und Rand - eine wichtige Qualitätsgröße. Sie

kann vor und hinter der Fertigstraße gemessen werden, aus technischen Gründen aber nicht zwischen den Walzgerüsten. Eine genaue Kenntnis ist jedoch aus produktionstechnischen Gründen wichtig. Der Stand der Technik ist die Berechnung dieser Zwischenprofile durch das wiederholte Durchrechnen eines mathematischen Modells des Walzvorganges für jedes Gerüst und eine ständige Anpassung von adaptiven Termen dieses Modells an die Messdaten.

Es wurde gezeigt, dass mit einem adaptiven neuronalen Netz, das mit Eingangs- und Ausgangsprofil sowie allen vorhandenen Kenn- und Stellgrößen trainiert wird, die Vorausberechnung des Endprofils mit deutlich höherer Genauigkeit vorgenommen werden kann. Das Problem ist, dass dieses Netz die Übertragungsfunktion der gesamten Straße repräsentiert, Zwischenprofile können nicht ausgegeben werden.

Daher wird der Versuch gemacht, beide Eigenschaften zu verbinden: Die genaue Endprofilmodellierung eines neuronalen Netzes wird kombiniert mit der Fähigkeit des iterierten Modells, Zwischenprofile zu berechnen. Dabei wird der in Form von Messdaten bekannte gesamte Prozess als iterierte Verknüpfung von technisch identischen Teilprozessen angesehen. Die Gewinnung eines Modells des Einzelprozesses entspricht damit der Berechnung der iterativen Wurzel des Gesamtprozesses.

Abkürzungen und Begriffe

Adaline	Adaptive Linear Network = Lineares Neuronales Netz
Bias	Additiver konstanter Eingang in ein Neuron
Layer	Schicht eines Netzes
iLayer	Iterative Layer = Teilnetz zur Iteration, ggf. aus mehreren Schichten
MLP	Multilayer Perzeptron = Mehrschichiges Perzeptron
RBF	Radiale Basis Funktionen = Netztyp
Mse	Mean Square Error = Mittlerer quadratischer Fehler
Rms	Root Mean Square error = Wurzel daraus
Sse	Sum Square Error = Summe der quadratischen Fehler

Inhaltsverzeichnis

Vorwort

Von einem konkreten Problem der industriellen Anwendung - der Modellierung von Zwischenprofilen in einem Stahlwalzwerk - auf ein fast 200 Jahre altes und in weiten Bereichen noch ungelöstes mathematisches Problem geführt zu werden, war eine erstaunliche Erfahrung.

Die „Quadratwurzel einer Funktion“ ist ein so überaus einfaches Konzept, dass es verwundern muss, wie wenig bekannt es selbst unter Mathematikern ist. Die Literatur zu diesem Gebiet ist sehr überschaubar: Ganze 100 Artikel und Bücher förderte eine Literaturrecherche zutage.

Seit Charles Babbages *„Essay towards the calculus of functions“* [1815] hat man zwar viele Einsichten zur Theorie der Funktionswurzeln gewonnen, die Anwendung zur Lösung praktischer Probleme wird hingegen immer noch durch kaum entwickelte Verfahren zu ihrer konkreten Berechnung behindert. Ein Übersichtsartikel aus dem Jahr 2001 beklagt: *„...one should not expect results on iterative roots in a general situation. In fact, even roots of polynomials are not described. Even worse: we do not know whether every complex cubic polynomial has a square root...“* [Baron 2001].

Allerdings wird die Kompliziertheit vielleicht einsichtig, wenn man bedenkt, dass das Finden von bestimmten Funktionswurzeln auch als „inverses Problem der Chaostheorie“ betrachtet werden kann. Dabei haben iterative Wurzeln faszinierende Anwendungen in den Grundlagen der Physik über Komplexitätstheorie bis hin zur Steuerung technischer Prozesse und in der Philosophie der Zeit.

Die hier vorgestellte Möglichkeit, das Problem der „iterativen Wurzeln“ auf neuronale Netze abzubilden und deren Lernfähigkeit dazu zu nutzen, zwar keine abstrakt formalen Lösungen, dafür aber numerische Näherungen auch bei rein datenbasierten Problemstellungen zu finden, zeigt, dass jenseits der bekannten Einsatzfelder auf neuronale Netze noch weitere überraschende Anwendungsbereiche warten.

Diese Arbeit entstand zum größten Teil während meiner Tätigkeit am FORWISS Erlangen, dem Bayerischen **For**schungszentrum für **Wiss**ensbasierte Systeme (www.forwiss.de), im Rahmen des Verbundprojektes AENEAS, ***A**nwendung und **E**ntwicklung **Ne**uronaler Verfahren zur **A**utonomen Prozess**s**teuerung*. Die Finanzierung erfolgte z. T. aus Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01 IN 505 B.

Mein herzlicher Dank gebührt zunächst Professor Peter Protzel, der während seiner Tätigkeit am FORWISS das Projekt initiiert hat, aus dem diese Arbeit hervorgegangen ist und der mich dann an der TU-Chemnitz geduldig weiter betreut hat. Meine Kollegen Achim Lewandowski, Michael Tagscherer und Reiner Holve waren stets eine grosse Quelle der Inspiration und Hilfe. Und am wichtigsten: Die Zusammenarbeit hat Freude gemacht! Besonderer Dank gilt auch den Korefferenten Professor Günter Radons und Professor Manfred R. Schroeder, der schon früher mehrere meiner Projekte wohlwollend begleitet hat. Meine Frau Angela bitte ich wegen der Monate, in denen ich praktisch rund um die Uhr am Computer verwurzelt war, um Vergebung. Und bei unserer Tochter Zoë Helene möchte ich mich für ihre Geduld und das perfekte Timing bedanken: Sie wartete bis zum Abend des Tages, an dem ich diese Arbeit ausgedruckt, gebunden und zum Prüfungsamt geschickt hatte, bis sie am 4. Juli 2001 zu uns auf die Welt kam!

KAPITEL 1

Einleitung

1.1 Erfahrung - Theorie - Anwendung

Das Verhältnis von Empirie, Theorie und Anwendung wird in verschiedenen Wissenschaftsgebieten oft unterschiedlich gewertet. Dem Theoretiker gilt die Theorie als das höherbewertete, das erstrebte Wissen. Der Experimentator kann jedoch mit der Evidenz eines einzigen Experimentes die schönste Theorie widerlegen. Und für den Ingenieur ist eine Theorie vor allem Mittel zum Zweck, eine bestimmte Anwendung zu ermöglichen.

Theorien sind Modelle der Wirklichkeit, die mindestens zwei Ziele verfolgen sollten: Zum einen die „Erklärung“ der empirischen Ergebnisse und zum anderen sollten sie Vorhersagen für bisher nicht bekannte Experimente ermöglichen, was auch die Voraussetzung für die Anwendbarkeit einer Theorie ist. Sie sind damit das klassische Bindeglied zwischen Erfahrung und Anwendung. Theorien, die keine nachprüfbaren Voraussagen machen, die „nicht falsifizierbar“ sind, werden heute nicht mehr ganz ernst genommen.

Wissensbasierte Verfahren versuchen, die Stufe der Theoriebildung zu überspringen, Anwendungen direkt aus den Daten zu realisieren ohne die Notwendigkeit, dass sie von Menschen „verstanden“ werden müssen. Die Theorie oder das „Modell“ wird automatisch erstellt und kann dem Anwender verborgen bleiben oder ihm als zusätzliches Ergebnis präsentiert werden.

Regressionsverfahren sind eine alte Methode, aus empirischen Daten Modelle zu erstellen, die zur Interpolation und Extrapolation verwendet werden können. In der Regel muss man jedoch ein Modell mit freien Parametern vorgeben; bekanntestes Beispiel ist die Methode der kleinsten Quadrate und die darauf aufbauende lineare oder nichtlineare Regression.

Künstliche Intelligenz in ihrer klassischen Zielsetzung sucht einen ähnlichen Weg durch formalisierte Methoden zu beschreiten: Aus Erfahrungen werden automatisiert

oder mit menschlicher Assistenz formale Regeln abgeleitet, die dann, besitzt der zugrundeliegende Kalkül die nötige Mächtigkeit, durch logische Verknüpfungen neue, in den Prämissen nicht direkt formulierte Aussagen erlauben. Der erstellte Satz an Regeln kann aber ggf. auch von Menschen verstanden und weiterentwickelt werden.

Die Verfahren der Neuroinformatik beschreiten einen anderen Weg. Verständnis im Sinne einer formalen Repräsentation des erlernten Wissens ist nicht beabsichtigt, von einem neuronalen Netz erwartet man nicht, dass es etwas „erklärt“, es soll nur „funktionieren“, das richtige Ergebnis liefern. Das Wissen ist dabei in diffuser Form in einer „Gewichtsmatrix“ gespeichert, die nur in ihrer Gesamtheit Bedeutung besitzt. Das einzelne Gewicht hat zwar einen technisch nachvollziehbaren Einfluss auf das Ergebnis, ist aber nicht sinnvoll in seiner Funktion zu isolieren.

Andererseits sind diese Verfahren letztlich auch durch eine mathematische Formulierung gegeben: Ein trainiertes Netz ist eine komplexe Funktion, die gegebene Eingangswerte auf eine oder mehrere Ausgangsgrößen abbildet. Das Training selbst ist nichts anderes als ein Regressionsverfahren, das die freien Parameter des Netzes - die Gewichte - so optimiert, dass der Fehler den diese Abbildung auf den Lerndaten macht, so klein wie möglich ist. Jedoch ist diese Ebene in der Regel nur für diejenigen interessant, die sich mit der Theorie der Netze selbst beschäftigen; in der Anwendung interessiert meist nur das Ergebnis.

Das bietet für den Anwender den Vorteil, dass aus vorhandenen Daten ohne eine Form von Theoriebildung ein Modell erzeugt werden kann. Die Netze sind gewöhnlich als Tools oder Softwarepakete vorhanden und können automatisiert mit Daten gefüttert werden. Die nötige Intelligenz, aus gegebenen Daten eine Prognose zu erstellen, wird damit nicht mehr dem Menschen abverlangt, sondern wohnt dem Verfahren inne.

Der Nachteil dabei ist allerdings, dass man wie gesagt kaum eine nachvollziehbare Erklärung bekommt, wie ein Ergebnis zustandekommt. Das Netz ist eine „Black Box“. Will man z.B. wissen, wie sich ein zusätzlicher Einfluss, der nicht in den Trainingsdaten vorhanden war, auf das Ergebnis auswirken würde, ist das Netz keine Hilfe. Auch für nur leicht abgewandelte Anordnungen erlischt in der Regel die Gültigkeit des neuronalen Modells.

Die Frage, inwieweit es doch möglich ist, mit neuronalen Methoden nur aus den Daten Wissen über die ihnen zugrundeliegenden Prozesse und Zusammenhänge zu gewinnen, ist ein aktuelles Forschungsthema, in das sich auch diese Arbeit einordnen lässt.

Für bestimmte Fragestellungen, die bisher eines theoretischen Modells zwingend bedurften, soll hier eine Methode aufgezeigt werden, wie allein aus Daten, die die Ein/Ausgangsbeziehung eines Systems beschreiben, ein Modell für dessen innere Struktur „automatisch“ gewonnen werden kann. Damit lassen sich dann auch bestimmte

Fragen nach anderen Zusammenhängen beantworten, als sie in den Trainingsdaten gegeben sind.

1.2 Fragestellungen

- Angenommen, wir haben ein Modell entwickelt, das den Umsatz eines Unternehmens im nächsten *Monat* aus schon jetzt bekannten Zahlen gut prognostizieren kann. Wie können wir daraus in Zeiten rasanter Wirtschaftsentwicklungen vorhersagen, wieviel nächste *Woche* umgesetzt wird?
- Angenommen, wir wissen aus 150 Jahren penibler Buchhaltung, wie sich die Populationen von Polarfüchsen und Schneehasen von *Jahr zu Jahr* verändert haben. Kann man daraus die Entwicklung der Populationen auch *monatsweise* bestimmen?
- Angenommen, wir wissen durch Messgeräte *vor* und *hinter* einer Walzstraße, wie Stahl, der *siebenmal* gewalzt wird, sich bei diesem Prozess verändert. Wie wirkt dann ein *einzelner* Walzvorgang?
- Angenommen, wir wissen durch periodische Messungen genau, wie der Zustand eines dynamischen Systems sich jeweils in einem *festen Zeitintervall* verändert. Können wir allein daraus die *kontinuierliche* Zeitentwicklung ableiten?
- Angenommen, wir wissen durch zwei Messgeräte im *festen Abstand*, wie sich die Geschwindigkeit eines Objektes, z.B. durch Reibung in einem Medium, von einen Messgerät zum anderen verändert. Können wir daraus die Geschwindigkeit auch an Orten *dazwischen* berechnen?
- Wir wissen, dass die Iteration der logistischen Gleichung zu immer komplexeren Funktionsverläufen bis hin zum Chaos führt. Ist aber die logistische Gleichung selbst vielleicht auch das Ergebnis der Iteration einer anderen, noch einfacheren Funktion?

Die klassische Methode, solche Fragen zu beantworten, ist es, eine Modellgleichung aufzustellen, z.B. eine Differentialgleichung und deren freie Parameter aus den Daten zu bestimmen. Ist damit eine genaue Reproduktion der empirischen Daten noch nicht möglich, muss die Gleichung modifiziert oder erweitert werden. Anschließend können mit diesem Modell dann auch die gestellten Fragen beantwortet werden.

Im Folgenden wird es darum gehen, ob und wie aus den Prämissen der Fragen auch ohne eine jeweils spezifische Theorie die gewünschten Ergebnisse abgeleitet werden können.

1.3 Diskrete und kontinuierliche Zusammenhänge

In der digitalen Welt machen die Zeit und andere Größen meist Sprünge, in der Physik wird sie als kontinuierlicher Fluss gesehen. Das Verhältnis zwischen diesen beiden Sicht- oder Beschreibungsweisen ist Gegenstand einer Reihe von Theorien und Methoden. Signal- und Informationstheorie sind für die Computertechnik zu zentralen Werkzeugen geworden. Begriffe wie Abtastrate, 16 oder 32 Bit Auflösung und Interpolation sind ins multimediale Wohnzimmer vorgedrungen.

Diskrete Zeit, fortschreitend in ganzzahligen Vielfachen von minimalen Taktzeiten, impliziert eine funktionsorientierte Beschreibungsweise von Prozessen oder Systemen, $x_{t+1} = f(x_t)$. Die Dynamik solcher Prozesse besteht dann aus der wiederholten Anwendung (Iteration) dieser elementaren Übertragungsfunktion f auf den Anfangszustand:

$$x_{t+n} = f^n(x_t) \tag{1.1}$$

Zeitkontinuierliche Systeme werden dagegen in der Regel durch Zeitentwicklungsgleichungen $x(t) = f(x_0, t)$ oder Differentialgleichungen beschrieben.

Messdaten liegen praktisch immer in zeitdiskreter Form vor. Kann aus Messungen des Zustands eines Systems zu diskreten Zeiten der kontinuierliche Zeitverlauf, also die Trajektorie im Zustandsraum, *exakt* rekonstruiert werden *ohne* Modellgleichungen aufzustellen? Oder kann entsprechend aus Messungen an *diskreten* Orten im Zustandsraum die *kontinuierliche* Bahnkurve bestimmt werden?

Ein altes, aber wenig bekanntes Problem aus dem Gebiet der Funktionalgleichungen liefert einen Ansatz hierzu, eine Brücke zwischen zeitdiskreten und zeitkontinuierlichen Systemen zu schlagen: Iterative Wurzeln oder fraktionale Iterationen.

1.4 Der Begriff der iterativen Wurzel

Für eine gegebene Funktion $f(x)$ betrachte man die Gleichung

$$g(g(x)) = f(x) \tag{1.2}$$

Eine Funktion $g(x)$, die Gleichung (1.2) löst, nennt man *Funktionswurzel* oder *iterative Wurzel* von $f(x)$.

Zu einer gegebenen Funktion $f(x)$ wird demnach eine Funktion $g(x)$ gesucht, die zweimal hintereinandergeschaltet gerade wieder $f(x)$ ergibt, wie folgende Beispiele demonstrieren:

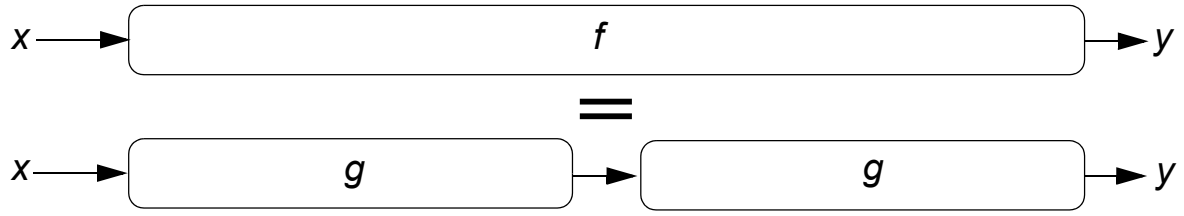

Abbildung 1.1 Schematisierung des Funktionswurzelbegriffes. Eine Funktion f ist so in zwei identische Teilfunktionen g zu zerlegen, dass das Ergebnis $y = f(x) = g(g(x))$ für alle x gleich bleibt.

- $f(x) = x + 2 \quad \Rightarrow \quad g(x) = x + 1$,

  denn $g(g(x)) = (x + 1) + 1 = x + 2 = f(x)$.

- $f(x) = 2x \quad \Rightarrow \quad g(x) = \pm\sqrt{2}\, x$,

  denn $g(g(x)) = \sqrt{2}(\sqrt{2}x) = 2x = f(x)$.

- $f(x) = x^2 \quad \Rightarrow \quad g(x) = |x|^{\sqrt{2}}$,

  denn $g(g(x)) = \left||x|^{\sqrt{2}}\right|^{\sqrt{2}} = x^2 = f(x)$.

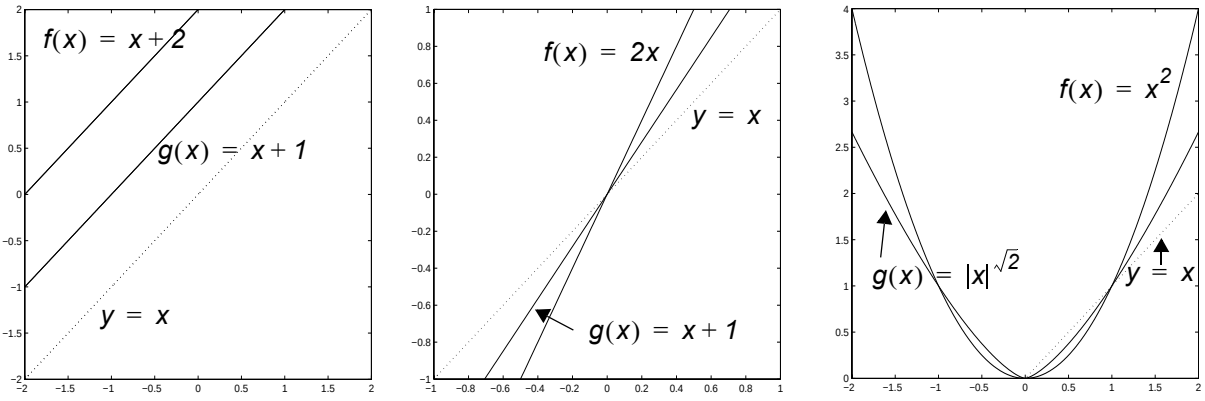

Abbildung 1.2 Die Graphen einiger Funktionen f und ihre iterativen Wurzeln g.

Im Gegensatz zur Differential- und Integralrechnung sind Rechenregeln für solche Funktionalgleichungen kaum bekannt. Der Leser möge sich selbst an den Funktionswurzeln von $f(x) = 2x + 1$ und $f(x) = x^2 + 1$ versuchen, um einen Eindruck von der *Andersartigkeit* dieses mathematischen Problems zu erhalten.[1]

1. Lösungen in Kapitel 2 und Anhang A

Das Finden von geschlossenen Lösungen für Funktionswurzeln kann äußerst schwierig, in vielen Fällen sogar unmöglich sein. Selbst die Frage, ob für eine gegebene Funktion überhaupt eine Wurzel existiert, ist nicht in allgemeiner Form geklärt [Humke 1989].

Die Literatur zu diesem Thema ist recht überschaubar und beschränkt sich im Wesentlichen auf sehr abstrakte Darstellungen der reinen Mathematik. Einfache Lösungshilfsmittel und numerische Methoden sind bisher kaum verfügbar. Ein Interesse scheint jedoch zu bestehen: In der Internet Newsgruppe alt.sci.math beispielsweise tauchen entsprechende Fragen regelmäßig auf und schon früh wurde von möglichen industriellen Anwendungen gesprochen [Kneser 1950].

Aufgrund der relativ geringen Anzahl von Arbeiten, die sich mit diesem Problem beschäftigen, und der trotzdem recht aufwendigen Literatursuche, sind im Literaturverzeichnis dieser Arbeit *sämtliche* gefundenen Artikel aufgeführt, die sich mit diesem Thema beschäftigen, selbst wenn hier nicht auf alle direkt eingegangen wird. Lesern, die sich tiefer in die Theorie der iterativen Wurzeln einarbeiten wollen, ist diese nahezu vollständige Bibliografie sicher willkommen.

Die praktischen Probleme, die auf diese Theorie zurückzuführen sind, beschränken sich allerdings nicht auf den Fall reellwertiger Funktionen, sondern verlangen u.a. folgende Erweiterungen:

1. x ist keine reelle Zahl, sondern ein n-dimensionaler Vektor.
2. Auch Wurzeln höheren Grades sind zu berechnen: $g(g(\ldots g(x)\ldots)) = f(x)$.
3. Falls keine Lösung existiert, ist zumindest die bestmögliche Annäherung zu finden.
4. Falls die Lösung nicht eindeutig ist, sollten die sinnvollen Ergebnisse erkannt und ausgewählt werden.
5. Es treten zusätzliche Eingangsgrößen auf, die bei der Verkettung der Wurzeln für jede Teilfunktion unterschiedlich sind, also durch die Form $f(x, p_1, p_2) = g(g((x, p_1), p_2))$ darstellbar sind.
6. $f(x)$ ist nicht explizit als algebraischer Ausdruck, sondern in Form einer Wertetabelle (x, y) aus Messdaten gegeben, die gegebenenfalls rauschbehaftet sein können.
7. Alle bekannten Probleme bei der Identifikation des Systems f können evtl. zusätzlich auftreten.

Während die mathematische Theorie der iterativen Wurzeln sich eingehend mit den Punkten 1-4 beschäftigt, sind bisher keine Ansätze bekannt, die die anderen gestellten Probleme angehen.

In dieser Arbeit werden Verfahren entwickelt, die, aufbauend auf neuronalen Netzen, die eine etablierte Methode zur Funktionsapproximation sind, auf einfache Weise auch approximative Lösungen von Funktionswurzeln berechnen. Vorausgesetzt, dass ein neuronales Netz f erfolgreich modellieren kann und überhaupt eine Lösung g existiert, kann auf diese Weise ebenfalls ein Modell für g erstellt werden.

1.5 Verallgemeinerung des Iterationsbegriffes

Die eben beschriebene lässt sich als direkte Übertragung des Wurzelbegriffes von Zahlen auf Funktionsräume verstehen. Grundlegend ist dabei der Begriff der Iteration einer Funktion: Sei $f(x)$ irgendeine Funktion, dann nennt man $f(f(x)) = f \circ f(x) = f^2(x)$ die *Iterierte* von f. f^n bedeutet in der Potenzschreibweise dann die n-te Iterierte von f. Des Weiteren ist f^{-1} eine gebräuchliche Notation der Inversen von f und f^{-n} der n-fach iterierten Inversen von f. Damit sind schon alle ganzzahligen Potenzen von allgemeinen Funktionen definiert.

Die Suche nach den Funktionswurzeln einer Funktion kann als das *inverse Problem* der Iteration angesehen werden oder als Verallgemeinerung der Potenzdarstellung auf *gebrochene* Exponenten oder Iterationstiefen: So, wie $\sqrt[n]{a} = a^{1/n}$ ist, schreibt man für die n-te Funktionswurzel von f auch $f^{1/n}(x)$ und verwendet die Bezeichnung *iterative Wurzel*.

Darüber hinaus kann man $f^{1/n}$ wiederum m-fach iterieren und schreibt das Ergebnis als *fraktionale Iteration* $f^{m/n}$. Dies erweitert die Potenzoperation bei Funktionen auf beliebige rationale Exponenten.

Vollzieht man noch den Übergang zu *reellen* Exponenten, nennt man die Menge aller Funktionen $f^t \quad t \in \mathbb{R}$ die *kontinuierliche iterative Semigruppe* von f, die die diskrete Menge $f^n \quad n \in Z$ in einen kontinuierlichen *Fluss* einbettet [Zdun 1977a].

1.6 Beispiel: Der freie Fall

Im Folgenden soll versucht werden, eines der elementarsten Probleme der Physik, den freien Fall, mit dessen mathematischer Beschreibung Galilei die moderne Physik begründet hat, im Kontext von Funktionalgleichungen zu formulieren und zu demonstrieren, welche Vorteile eine solche Beschreibung, besonders in experimentellen Zusammenhängen haben kann.

1.6.1 Iteration des Ortes

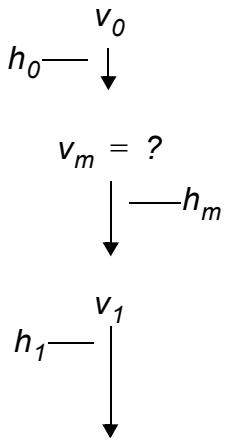

Abbildung 1.3 Ein Objekt fällt von h_0 bis h_1

Versuchsaufbau: Ein Objekt fällt von Höhe h_0 auf Höhe h_1. Bei h_0 hat es die gegebene oder gemessene Geschwindigkeit v_0, bei h_1 misst man v_1. Das Experiment: Wir variieren die Anfangsgeschwindigkeit v_0 und messen die jeweils resultierende Endgeschwindigkeit v_1. Diesen Zusammenhang nehmen wir als Messkurve auf. Frage: Welche Geschwindigkeit hat das Objekt am Ort h_m in der Mitte zwischen h_0 und h_1? - Oder noch allgemeiner: Wie ist die Geschwindigkeit an einem beliebigen Ort zwischen h_0 und h_1 oder jenseits dieser Punkte? Können diese Fragen allein aus den Ergebnissen des Experimentes beantwortet werden?

Mit Kenntnissen von der Physik des freien Falls und vorausgesetzt, es gibt keine „Störeinflüsse“ wie Reibung oder Inhomogenitäten des Gravitationsfeldes, ist es leicht, eine Gleichung für diese Fragen zu finden: Beispielsweise erhält man mit Hilfe des Energiesatzes (mit Erdbeschleunigung a) den Zusammenhang

$$\frac{1}{2}mv_0^2 + ma(h_1 - h_0) = \frac{1}{2}mv_1^2,$$

also ist bei festem $\Delta h = h_1 - h_0$ die Geschwindigkeit v_1 eine Funktion von v_0

$$v_1 = f(v_0) = \sqrt{v_0^2 + 2a\Delta h}\,. \tag{1.3}$$

Eine Tabelle aus Messungen von v_0 und v_1 definiert diese Funktion exemplarisch und kann zur Verifikation des Fallgesetzes verwendet werden.

Allerdings gehen die Gleichungen vom freien Fall ohne Reibung aus. Die Fallgleichungen werden wesentlich komplizierter, wenn man Luftreibung mit einbezieht. Die Physik stellt dann Näherungen zur Verfügung, die die Reibung als Kraft proportional zur Geschwindigkeit (bei hoher Viskosität und langsamer Geschwindigkeit), dem Quadrat der Geschwindigkeit (Luftreibung) oder noch höheren Potenzen (in der Nähe der Schallgeschwindigkeit) mit einbeziehen. Diese Gleichungen enthalten aber in der Regel experimentell zu bestimmende Konstanten. Auch ist die Integration der Bewegungsgleichungen bei einer Mischform aus diesen verschiedenen Reibungsarten sehr schwierig oder ist geschlossen gar nicht mehr möglich.

Aber eine Messung der Endgeschwindigkeit für viele Anfangsgeschwindigkeiten enthält ja alle diese Effekte und könnte dazu verwendet werden, die Koeffizienten der Fallgleichung *mit* Reibung zu bestimmen. Allerdings: Diese Gleichungen bleiben Näherungen. Die genauen Prozesse, die bei der Reibung von *realen* Objekten mit

realen Gasen auftreten, sind nur außerordentlich schwer in einfachen Gleichungen formulierbar, fast immer sind hier genaue Messungen erforderlich.

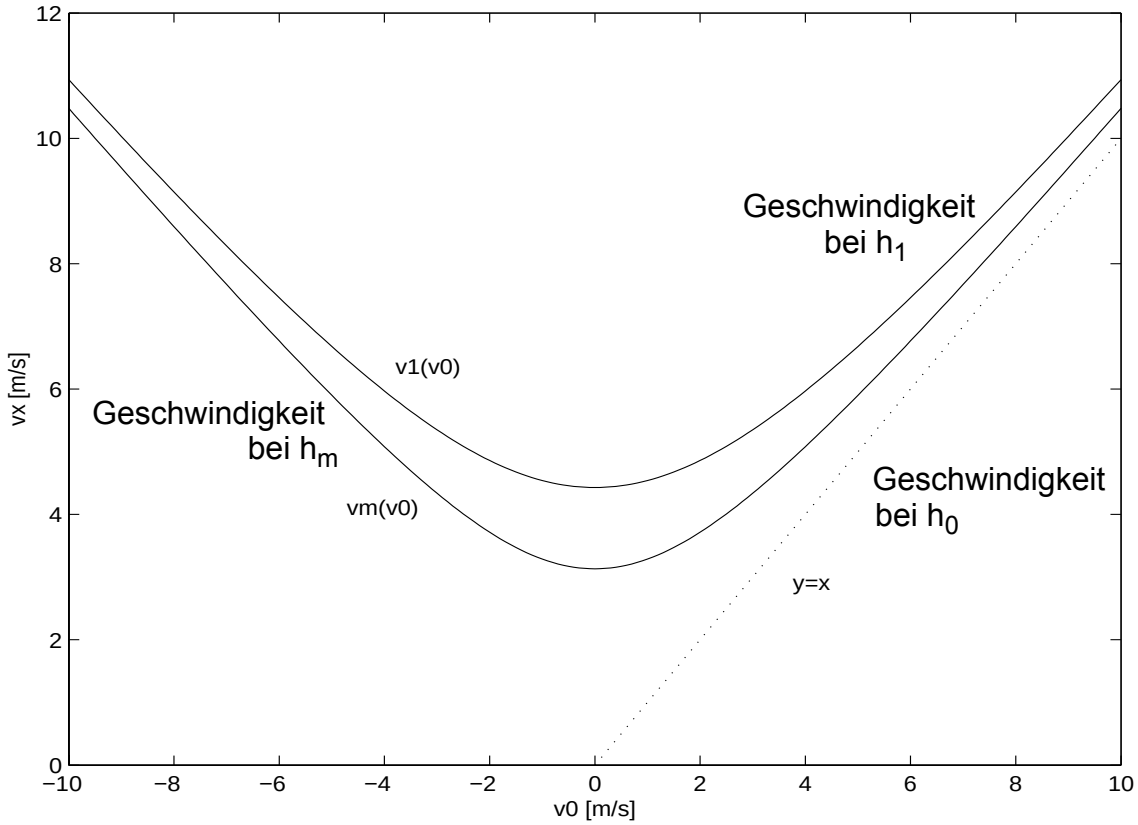

Abbildung 1.4 Abhängigkeit der End- von der Anfangsgeschwindigkeit. Fallhöhe: 1m (v_1) bzw. 1/2 m (v_m). Die Funktion $v_m(v_0)$ ist gleichzeitig die iterative Wurzel von $v_1(v_0)$.

Die Messwerttabelle v_0 in Abhängigkeit von v_1 und die dadurch implizit definierte Funktion $v_1 = f(v_0)$ ist (bei einem gut durchgeführten Experiment) oft die beste verfügbare Beschreibung der Realität. Frage ist: Kann man auch ohne eine „exakte mathematische“ Formulierung, sprich Fallgesetze und Reibungsgesetze, zu kennen, ein Modell erstellen, das den reibungsbehafteten Fallprozess modelliert, also (nachprüfbare) Voraussagen über die Geschwindigkeit an anderen Orten als h_1 macht und zwar mit einer Genauigkeit, die von der gleichen Größenordnung wie die Messungen ist?

Iterative Wurzeln erlauben genau das. Setzt man voraus, dass die Physik im Raumbereich h_0 bis $h_m = h_0 - \Delta h/2$ die gleiche ist, wie im Bereich h_m bis h_1, dann ist bei gegebener Anfangsgeschwindigkeit v_0 die Geschwindigkeit $v_m = g(v_0)$ berechenbar durch die Funktionswurzel von f

$$v_1 = f(v_0) = g(v_m) = g(g(v_0)) .$$

Im reibungsfreien Fall kann man das leicht verifizieren. Hier müsste die Funktion für die Geschwindigkeit v_m, die sich aus Gleichung (1.3) durch Halbieren von Δh ergibt

$$g(v_0) = \sqrt{v_0^2 + 2a(\Delta h/2)}$$

gerade die Funktionswurzel von

$$f(v_0) = \sqrt{v_0^2 + 2a\Delta h}$$

sein. Dies ist leicht nachprüfbar durch Iteration von g:

$$g^2(v_0) = \sqrt{\sqrt{v_0^2 + a\Delta h}^2 + a\Delta h} = \sqrt{v_0^2 + 2a\Delta h} = f(v_0)$$

Wahrscheinlich wäre es ohne physikalisches Vorwissen nicht so leicht gewesen, die iterative Wurzel einer Funktion der Form $f(x) = \sqrt{x^2 + c}$ zu bestimmen. Haben wir nur die implizite Darstellung von f durch eine Messwertetabelle, ist auf jeden Fall ein numerisches Verfahren notwendig.

Besonders intuitiv wird dieser Formalismus, wenn man ohne Beschränkung der Allgemeinheit $h_0 = 0$ und $h_1 = 1$ setzt. Dann ist

$$v_1 = f(v_0) = \sqrt{v_0^2 + 2a}$$

und die allgemeine Funktion für die Geschwindigkeit v bei gegebener Anfangsgeschwindigkeit v_0 in der Höhe h

$$v(v_0, h) = \sqrt{v_0^2 + 2ah} = f^h(v_0).$$

Das wiederum entspricht der Einbettung der Funktion f in ihre *kontinuierliche iterative Semigruppe* $f^h \; h \in \mathbb{R}$. Dabei ist f^0 die Identität, $f^1 = f$, $f^n \; n \in \mathbb{N}$ die n-te Iterierte von f, $f^{1/n}$ die n-te iterative Wurzel von f und $f^{m/n}$ eine fraktionale Iteration von f, die ja eine beliebig genaue Näherung für jedes reelle h ermöglicht. f^{-h} ist die Inverse von f^h und beschreibt die Geschwindigkeit bei negativem h, sofern existent und nicht unbedingt eindeutig.

Das bedeutet, wenn man in der Lage ist, die kontinuierliche Iteration von f zu bestimmen, kann man den kompletten Geschwindigkeitsverlauf des Falles berechnen, ohne explizite Fallgesetze zu verwenden.

Dieses Verfahren funktioniert unter der Voraussetzung, dass die Physik entlang der Fallrichtung invariant ist. Die Messwerttabelle muss also eindeutig sein: Für unterschiedliche h_0 muss bei gleichem v_0 und Δh stets das gleiche v_1 herauskommen. *Aus der Kenntnis, wie sich eine Größe zwischen zwei diskreten Orten im festen Abstand transformiert sowie der Annahme räumlicher Isotropie, ist durch die verallgemeinerte Iteration dieser Transformation auch der kontinuierliche räumliche Verlauf dieser Größe bestimmt.*

Verwendet man ein numerisches Verfahren zur Bestimmung der Iteration, ist es dabei *nicht* notwendig, ein geschlossen formuliertes Modell aufzustellen. Allein aus den Messdaten von zwei Orten können die Werte an anderen Orten mit ähnlicher Genauigkeit bestimmt werden. Und dies ist nicht eingeschränkt auf den idealisierten Fall ohne Reibung, sondern ist - solange es keinen von h abhängigen Einfluß auf das fallende Objekt gibt - allgemein gültig, sogar im relativistischen Fall.

1.6.2 Zeitentwicklung als verallgemeinerte Iteration

Versuch: Ein Körper fällt wieder im freien Fall und wir messen seine Geschwindigkeit diesmal zu zwei *Zeit*punkten t_0 und t_1 im festen Abstand Δt. Dadurch ist auch wieder eine Funktion $v_1 = f(v_0)$ definiert.

Die Messungen sollten natürlich ergeben, dass

$$v_1 = v_0 + a\Delta t$$

ist. Die Funktionswurzel von

$$f(v) = v + a\Delta t$$

ist, wie man leicht sieht,

$$f^{1/2}(v) = v + \frac{1}{2}a\Delta t$$

und die verallgemeinerte Iterierte ist

$$f^t(v) = v + ta\Delta t.$$

Das wenig erstaunliche, aber mit der Erfahrung konsistente Ergebnis, das diese Methode liefert, besagt somit, dass die Geschwindigkeit während des Falls linear ansteigt.

Dies lässt sich auf die Zeitentwicklung der Höhe nicht ohne weiteres übertragen: Eine Messserie, die die Höhe zu zwei Zeitpunkten im festen Abstand misst, wird keine eindeutige Beziehung zwischen h_0 und h_1 feststellen, unterschiedliche Anfangsgeschwindigkeiten führen bei gleicher Anfangshöhe zu unterschiedlichen Endpunkten.

Daher wählt man eine vollständige Darstellung des Systems im Zustandsraum: Es wird beschrieben durch die Angabe von h und v, die als Vektor geschrieben werden können. Der Zustand des Systems ändert sich vom Anfangszustand (v_0, h_0) während der Zeit Δt zu $(v_1, h_1) = f(v_0, h_0)$, f ist jetzt eine Abbildung $\mathbb{R}^2 \to \mathbb{R}^2$. Aus den Fallgleichungen wissen wir, wie f aussieht:

$$(v_1, h_1) = f(v_0, h_0) \quad \text{mit} \quad \begin{aligned} v_1 &= v_0 + a\Delta t \\ h_1 &= h_0 + v_0\Delta t + \frac{1}{2}a\Delta t^2 \end{aligned} \tag{1.4}$$

Zum mittleren Zeitpunkt $t_m = t_0 + \frac{\Delta t}{2}$ sollte wieder gelten

$$(v_1, h_1) = g(v_m, h_m) = g(g(v_0, h_0)) = f(v_0, h_0)$$

Setzen wir für g die Funktion

$$(v_m, h_m) = g(v_0, h_0) \text{ mit } \begin{aligned} v_m &= v_0 + \frac{1}{2}a\Delta t \\ h_m &= h_0 + \frac{1}{2}v_0\Delta t + \frac{1}{8}a\Delta t^2 \end{aligned} \tag{1.5}$$

ein, kann man wieder leicht sehen, dass man das Gleichungssystem (1.4) durch Iteration von (1.5) erhält, g also eine iterative Wurzel von f ist. Der Übergang zur kontinuierlichen Zeit erfolgt, wie zuvor demonstriert, durch die kontinuierliche bzw. fraktionale Iteration von f.

Diese Beispiele zeigen die Mächtigkeit der verallgemeinerten Iteration: Kennt man die Funktion, die die Transformation des Zustandes eines Systems während eines *fixen* Zeitintervalls allgemein beschreibt, so lässt sich daraus die *kontinuierliche* Zeitentwicklung ableiten.

Besonders interessant wird dieses Verfahren, wenn keine Theorie vorliegt, sondern lediglich Messwerte zu diskreten Zeitpunkten gegeben sind, die implizit eine Funktion definieren. Auch dann ist es möglich, daraus die kontinuierliche Zeitentwicklung mit einem numerischen Verfahren zu rekonstruieren.

1.7 Modellierung industrieller Prozesse

Konkreter Anlass für diese Studie war das Problem, bestimmte Eigenschaften von Stahlblechen zu modellieren, die eine Walzstraße durchlaufen. Diese Anlage besteht aus *mehreren technisch identischen* Walzgerüsten, die das Blech nacheinander passiert. Billiganlagen bestehen oft nur aus *einem* Gerüst und das Blech wird entsprechend oft hin- und hergewalzt. Am Anfang und am Ende des Prozesses können Eigenschaften wie Dicke oder Profil des ein- und auslaufenden Stahlbandes durch Sensoren bestimmt werden. Für die Prozesssteuerung sind allerdings auch die Zwischenwerte interessant. Da jedoch Messungen zwischen den Gerüsten aus technologischen Gründen nicht möglich sind, ist ein *Soft Sensor* nötig, der die gewünschten Daten anhand eines Modells ermittelt.

Mit Mitteln der adaptiven Systemidentifikation ist es möglich, messdatengetrieben ein sehr gutes Modell für das Verhalten der gesamten Walzstraße zu erstellen. Daraus

ein Modell für das einzelne Gerüst zu gewinnen, entspricht der Berechnung von etwas Ähnlichem, wie der iterativen Wurzel der Anlage. Das wiederholte Durchrechnen dieses Teilmodells erlaubt dann, die Entwicklung des Walzgutes zwischen den einzelnen Anlagenteilen zu modellieren.

Abbildung 1.5 Ein iterierter Prozess im Stahlwerk. In diesem Walzwerk durchläuft Stahlblech sieben baugleiche Walzgerüste in Folge. Foto Copyright SIEMENS AG

Allgemein lautet die Problemstellung: *Der Zusammenhang von Eingangs-, Ausgangs- und Stellgrößen eines Prozesses ist bekannt oder messbar, außerdem weiß man, dass dieser Prozess auf der wiederholten Anwendung des gleichen Vorganges evtl. mit jeweils anderen Einstellungen beruht. Die Modellierung des Teilprozesses entspricht dann einer iterativen Wurzel des Gesamtprozesses.*

1.8 Interpolation von Zeitreihen

Beispiel sei ein Modell für Umsatzverläufe eines Unternehmens. Es liegen große Mengen an bisher *monatlich* gesammelten relevanten Daten vor, aus denen man ein gutes Modell *M* zur Prognose des jeweils nächsten Monats gewinnen kann. Gewünscht ist jedoch nun ein Modell für den Umsatz der nächsten *Woche*n. Man muss also das Monatsmodell interpolieren. Im einfachsten Fall nimmt man die Monatsvorhersage und teilt die Änderung durch 4. Bei nichtlinearen Zusammenhängen ist dies aber in der Regel nicht die optimale Lösung. Korrekt wäre die Bestim-

mung der fraktionalen Iterationen des Monatsmodells $M^{1/4}$, $M^{2/4}$ und $M^{3/4}$, um den Verlauf in den ersten 3 Wochen des Monats zu modellieren.

Allgemein setzt Interpolation immer ein Modell voraus. Lineare, bikubische oder Splineinterpolationen sind weit verbreitete Methoden. Fouriertransformation und anschließende Rücktransformation mit doppelter Abtastrate ist in der Signalverarbeitung üblich. Ist jedoch der Zusammenhang, der das Aufeinanderfolgen von Zuständen beschreibt, bekannt, dann ist die iterative Wurzel dieses Modells die beste Approximation von Zwischenwerten.

1.9 Iterierte Abbildungen und dynamische Systeme

Üblicherweise werden dynamische Systeme durch Differentialgleichungen beschrieben, in denen die Zeit als Parameter auftaucht. Ist die Differentialgleichung eines Systems bekannt, besteht auch kein Bedarf an Interpolationsverfahren, da sie dann bis zu jedem beliebigen Zeitpunkt integriert werden kann.

Aus diskreten Daten gewonnene Modelle liegen jedoch meist in Form eines impliziten iterierten Funktionszusammenhanges vor.

Iterierte Abbildungen spielen in der nichtlinearen Mathematik und Chaostheorie eine zentrale Rolle. Die übliche Betrachtungsweise geht von einer iterierten Abbildung aus, sucht nach Fixpunkten, Attraktoren, Bifurkationen etc. Die Methode der fraktionalen Iteration erlaubt ganz neue Herangehensweisen und Fragestellungen:

- Ist die Abbildung selbst die Iterierte einer anderen, evtl. einfacheren Abbildung und wie sieht diese aus?
- Kann eine iterierte Abbildung sogar auf einen kontinuierlichen Prozess zurückgeführt werden bzw. in ihn eingebettet werden?

Dies ist äquivalent zur Frage nach Existenz und Aussehen von iterativen Wurzeln. Bei Systemen, die durch Differentialgleichungen definiert sind, z.B. bei dem Lorenz System, ist a priori gegeben, dass sie „einbettbar“ sein müssen. Aus ihnen durch „Abtastung“ erzeugte iterative Abbildungen sind natürlich zerlegbar und besitzen beliebige iterative Wurzeln.

Bei diskreten Systemen dagegen, wie z.B. bei der logistischen Gleichung im Reellen (Feigenbaum) oder Komplexen (Mandelbrot), sind die oben formulierten Fragen allerdings keineswegs klar. Beide Systeme können *nicht* iterative Wurzeln jeden beliebigen Grades haben, denn zeitkontinuierliche Systeme in ein- oder zweidimensionalen Räumen können nach dem Satz von Poincaré-Bendixon kein chaotisches Verhalten zeigen. Die Frage aber, ob sie überhaupt keine iterativen Wurzeln besitzen, ist nicht trivial.

KAPITEL 2

Iterative Wurzeln und Fraktionale Iteration

Dieses Kapitel stellt einige mathematischen Grundlagen aus der Iterationstheorie zusammen. Ausgehend von der *Iteration* einer Funktion wird dieser Begriff auf nicht-ganzzahlige, die sogenannten *rationalen Iterationen* erweitert, was dann zur Idee der *Funktionswurzel* führt. Neben der abstrakt mathematischen Darstellung ist die Interpretation der Iteration als Funktion der Zeitentwicklung in zeitdiskreten dynamischen Systemen eine intuitive Anwendung.

2.1 Iteration einer Funktion

2.1.1 Definitionen

Sei X eine beliebige Menge und $\mathfrak{I}(X)$ die Menge aller Abbildungen von X auf sich selbst. Die natürliche innere Operation auf solchen Selbst-Abbildungen $f, g, h \in \mathfrak{I}(X)$ ist die Komposition oder Hintereinanderschaltung, $f \circ g$. Sie ist assoziativ, $(f \circ g) \circ h = f \circ (g \circ h)$, aber in der Regel nicht kommutativ, $f \circ g \neq g \circ f$. $\mathfrak{I}(X)$ bildet mit dem Verkettungsoperator „$\circ$“ eine Semigruppe, die Identität ist id_X.

Die Potenzen $f^n, n \in \mathbb{N}_0$ eines Elementes $f \in \mathfrak{I}(X)$ bezeichnet man als *Iterationen* oder *Iterierte* von f:

$$f^0 = id_x, \quad f^1 = f, \quad f^{n+1} = f \circ f^n, \qquad n \in \mathbb{N}_0$$

Die Iterationen einer Abbildung f bilden eine kommutative Semigruppe und es gilt

$$f^m \circ f^n = f^n \circ f^m = f^{n+m}, \qquad n, m \in \mathbb{N}_0. \tag{2.1}$$

Wenn $f: X \to X$ eine umkehrbare Abbildung ist, kann man auch *negative* Iterationen definieren. Setzt man zunächst formal in Gleichung (2.1) $m = -1$ und $n = 1$, erhält man

$$f^{-1} \circ f = f^{-1+1} = f^0 = id.$$

Dies entspricht genau der Definition eines inversen Elements, f^{-1} ist die Umkehrfunktion von f. Weiter ist f^{-n} mit $n = k+1$ rekursiv definiert durch

$$f^{-k-1} = f^{-k} \circ f^{-1}, \quad k \in \mathbb{N}_0.$$

Man kann auch sagen, die $-n$-te Iterierte ist die Umkehrfunktion der n-ten Iterierten:

$$f^{-n} = [f^n]^{-1}$$

Soweit sind dies allgemein bekannte Begriffe. Die Erweiterung auch auf nicht-ganzzahlige Iterationen, die im Folgenden durchgeführt werden soll, ist dagegen recht ungeläufig.

2.1.2 Bemerkungen zur Schreibweise

Die hier verwendete Exponentialschreibweise für Iterationen kann zu Uneindeutigkeiten mit der Potenzierung in X führen. Daher wird in dieser Arbeit folgende Notation vereinbart: Bei Funktionen und Abbildungen bedeuten f^2, $f^2(x)$ oder $[f(x)]^2$ stets die Iteration $f \circ f$ bzw. $f(f(x))$. Falls dagegen die algebraische Multiplikation von Funktionen bzw. Funktionsergebnissen gemeint ist, $f(x)^2 = f(x) \cdot f(x)$ wird dies im Text ausdrücklich betont werden. Bei Variablen $x \in X$ ist dagegen nur $x^2 = x \cdot x$ sinnvoll. f, g bezeichnen bekannte Funktionen, φ und ϕ gesuchte Funktionen.

Bei den Iterationsindces wird die Konvention eingehalten, dass m, n ganze Zahlen bedeuten, r rationale und t reelle.

2.1.3 Einige Begriffe aus der Iterationstheorie

Die unendliche Folge $f^n(x), n = 0, 1, 2, ..., \infty$ wird in der mathematischen Literatur als *f-Splinter* von x bezeichnet. Zentraler Begriff der Iterationstheorie ist die Äquivalenzklasse der *Orbits* einer Funktion f:

Die Punkte $x, y \in X$ nennt man äquivalent unter der Iteration f, wenn es Zahlen $m, n \in \mathbb{N}_0$ gibt, so dass $f^m(x) = f^n(y)$. Sie gehören zum gleichen Orbit: $x, y \in \Omega$.

Die Orbitstruktur einer gegebenen Abbildung f kann zur vollständigen Repräsentation von f verwendet werden. Sie ist die Menge aller Orbits von f, wobei die Schnittmenge zweier Orbits leer ist und die Vereinigung aller Orbits X ergibt.

Alle Punkte eines Orbits landen also bei wiederholter Iteration irgendwann bei mindestens einem gemeinsamen Punkt: Die Schnittmenge ihrer Splinter ist nicht leer. Die Menge der Punkte, die zu *allen* Splintern eines Orbits gehören, wird *Zyklus* des Orbits genannt. Die Anzahl der Elemente eines Zyklus wird Länge des Zyklus genannt. Jeder Orbit hat höchstens einen Zyklus.

x ist ein Fixpunkt von f, wenn $f(x) = x$, Elemente eines Zyklus sind entweder Fixpunkt von f, oder Fixpunkt von f^m.

Besonders einfach wird die Darstellung, wenn X endlich ist, d.h. $n \in \mathbb{N}$ Elemente besitzt. Dann lassen sich Abbildungen $X \Rightarrow X$ als nxn-Matrizen schreiben, die in jeder Zeile genau eine 1, sonst überall 0 enthalten.

2.1.4 Interpretation der Iteration als Zeitentwicklung

Die Theorie iterierter Funktionen, die auf diesen Definitionen und Begriffen aufbaut, ist gerade in den letzten Jahren zu enormer Bedeutung gelangt. Gebiete wie Synergetik, dynamische Systeme oder deterministisches Chaos basieren zu einem wichtigen Teil auf iterierten Funktionssystemen.

Sei $X \subset \mathbb{R}^n$ der Zustandsraum eines autonomen Systems, d.h. es wirken keine äußeren (veränderlichen) Einflüsse. Jedes Element $x \subset X$ ist also ein möglicher Zustand. $f(x)$ sei die Funktion, die beschreibt, wie sich das System in einem normierten festen Zeitschritt ($\Delta t = 1$, ohne Beschränkung der Allgemeinheit) ändert.

Ist das System zum Zeitpunkt $t = 0$ im Zustand x_0, so ist es bei $t = 1$ im Zustand $f(x_0)$ und zum Zeitpunkt $t = n$ im Zustand $f^n(x_0)$.

Die Iteration beschreibt somit die Zeitentwicklung eines zeitdiskreten Systems, f^t ist der diskrete Zeitentwicklungsoperator [Targonski 1981].

Die mögliche Einbettung dieses zeitdiskreten Systems in eine kontinuierliche Zeit führt auf das Konzept der nicht-ganzzahligen Iterationen. Eine solche Einbettung der Semigruppe der „natürlichen“ Iterationen $f^n(x)$, $n \in \mathbb{N}_0$ in eine „kontinuierliche iterative Semigruppe“ $F(x, t)$, $t \in \mathbb{R}$ ist möglich, wenn

$$F(x, n) = f^n(x) \text{ für alle } n \in \mathbb{N}_0$$

und F der Translationsgleichung

$$F(F(x, t_1), t_2) = F(x, t_1 + t_2)$$

für beliebige t genügt [Zdun 1977b]. Anstatt $F(x, t)$ schreibt man dann formal $f^t(x)$ für die *kontinuierliche Iteration* von f.

Jedes zeitkontinuierliche System lässt sich durch „Abtastung“ mit fester Abtastrate Δt in ein zeitdiskretes abbilden. Umgekehrt ist es nicht möglich, jedes beliebige zeitdiskrete System in ein zeitkontinuierliches einzubetten. Nach dem Satz von Poincaré Bendixon gilt z. B., dass kontinuierliche Dynamiken im $\mathbb{R}^1$ und $\mathbb{R}^2$ entweder asymptotisch stationäre Fixpunkte oder periodische Bahnen haben [Kroll 1983]. Sehr bekannte Beispiele für iterative Abbildungen, die diese Bedingung nicht erfüllen, sind z. B. die Mandelbrot'sche Abbildung $z_{n+1} = z_n^2 + c$ und die Logistische Gleichung $x_{n+1} = cx_n(1 - x_n)$, die bekanntlich für bestimmte Wertebereiche von c chaotisches Verhalten zeigen und folgerichtig nicht einbettbar sein sollten.

Eine zusätzliche Forderung für physikalische Systeme ist die Stetigkeit von F in t, ein System springt nicht instantan von einem Zustand in einen anderen.

Eine notwendige Bedingung für diese Einbettung ist die Existenz *iterativer Wurzeln* beliebigen Grades der Funktion $f(x)$ [Targonski 1981]. Zur Berechnung eines kontinuierlichen Zeitverlaufes reicht in der Regel die Berechnung der *rationalen Iterationen* von f.

2.2 Iterative Wurzeln und rationale Iterationen

2.2.1 Definition

Eine Funktion φ, die die Funktionalgleichung

$$\varphi^n = f \tag{2.2}$$

für $f \in F(X)$ löst, bezeichnet man als n-te *Funktionswurzel* oder n-te *iterative Wurzel* von f und schreibt in Erweiterung der Exponentialschreibweise für die Iteration

$$\varphi = f^{1/n}$$

Lösungen der Funktionalgleichung

$$\varphi^n = f^m$$

mit $m \in \mathbb{N}_0$, $n \in \mathbb{N}$ bezeichnet man als *rationale* oder *fraktionale Iterationen* von f und schreibt

$$\varphi = f^{m/n}.$$

So wird die Notation der Iteration auf rationale Exponenten $r \in Q$ erweitert.

Negative Exponenten bedeuten auch hier die fraktionale Iteration der Umkehrfunktion f^{-1} oder die Umkehrfunktion der fraktionalen Iterierten, falls die Inversen existieren.

2.2.2 Kontinuierliche Iteration

Die fraktionalen Iterationen $f^r(x)$, $r \in Q$ haben eine innige Verbindung zur kontinuierlichen iterativen Semigruppe $f^t(x)$, $t \in \mathbb{R}$ [Zdun 1985]. Diese wird auch als kontinuierliche Iteration einer Abbildung bezeichnet

Insbesondere gilt, dass ein zeitdiskretes System mit der Zeitschrittfunktion f nur dann in eine kontinuierliche Zeit eingebettet werden kann, wenn iterative Wurzeln $f^{1/k}$ für beliebiges $k \in \mathbb{N}$ existieren.

Des Weiteren sind iterative Wurzeln hilfreich bei der konkreten Berechnung der (kontinuierlichen) Trajektorie $x(t)$ eines Punktes x_0, wenn nur die Zeitschrittfunktion f bekannt ist. Idealerweise sucht man einen geschlossenen Ausdruck der Form

$$x(t) = F(x_0, t).$$

Diesen kann man jedoch im Allgemeinen nicht angeben. Kennt man aber die n-te Wurzel von f und ist $t \approx m/n$, kann man die fraktionale Iteration $f^{m/n}$ als praktische Lösung ansehen, da die rationalen Zahlen dicht in $\mathbb{R}$ sind.

Zu beachten ist dabei allerdings doch ein Unterschied zwischen der kontinuierlichen Iteration f^t, betrachtet an den Zeitpunkten $t = 1/n$ und den iterativen Wurzeln $f^{1/n}$.

An zeitkontinuierliche Systeme wird gewöhnlich auch die Forderung nach Stetigkeit gestellt, d.h. für $\Delta t \to 0$ sollte $f^{t+\Delta t}(x)$ gleichmäßig gegen f^t konvertieren. Dies ist nicht immer für alle Lösungen von $\varphi^n = f$ der Fall. Nicht alle Lösungen φ von $\varphi^m = f^n$ sind Elemente der kontinuierlichen iterativen Semigruppe von f.

2.3 Eigenschaften von iterativen Wurzeln

Im Falle allgemeiner Mengen X und innerer Abbildungen kann man Folgendes zeigen: Sei $\varphi: X \to X$ eine Lösung der Gleichung $\varphi^n = f$.

φ ist surjektiv (bzw. injektiv, bijektiv) dann und nur dann, wenn f surjektiv (bzw. injektiv, bijektiv) ist.

Hat φ einen Fixpunkt an der Stelle x_a, so gilt dies genauso für f.

Für monotone Funktionen gilt weiter:

wenn der Grenzwert existiert, ist $\lim_{n \to \infty} f^n(x) = x$. [Hamilton 1946].

2.3.1 Existenz

Für den allgemeinsten Fall von Selbstabbildungen beliebiger Mengen fand Zimmermann [1978] eine Anzahl von Theoremen, die Aussagen zur Existenz von iterativen Wurzeln machen. Laut Targonski [1980] kann man damit die Gleichung $\varphi^n = f$ in all-

gemeiner Form als gelöst betrachten. Dabei wird von der natürlichen Iteration von f ausgegangen und f durch ihre Orbitstruktur beschrieben.

Definition: Seien n ein Teiler von $N \geq 2$ und $\Omega_1, \ldots, \Omega_n$ Orbits von f. Setze $\Omega = \bigcup_{i=1}^{n} \Omega_i$. Man nennt dann die Orbits $\Omega_1, \ldots, \Omega_n$ N-verbindbar, vorausgesetzt, es gibt eine Abbildung $\varphi_\Omega : \Omega \to \Omega$ für die $\varphi_\Omega^N = f|\Omega$ ist und φ_Ω genau einen Orbit hat.

Satz: $\varphi^n = f$ hat genau dann mindestens eine Lösung φ, wenn die Menge aller Orbits von f eine Dekomposition in eine Menge disjunkter Klassen zulässt, so dass die Kardinalität jeder Klasse ein Teiler von n ist und die Elemente jeder Klasse n-verbindbar sind.

Damit ist das Problem der Existenz von iterativen Wurzeln auf das Problem der Verbindbarkeit von Orbits reduziert, was jedoch die Schwierigkeiten bei der Lösung praktischer Anwendungen kaum vermindert. Immerhin konnte in einigen Fällen daraus die Nichtexistenz von iterativen Wurzeln bestimmter Abbildungen abgeleitet werden.

Ein weiterer Satz, mit dem häufig die Existenz einer Funktionswurzel von f ausgeschlossen werden kann, lautet:

Sei für irgendwelche $a, b \in X$ $f(a) = b$ und $f(b) = a$, $a \neq b$. Wenn dann für irgendein $x \in X$ aus der Beziehung $f^2(x) = x$ folgt, dass $x \in \{a, b, f(x)\}$ sein muss, hat die Gleichung $\varphi^2 = f$ keine Lösung [Kuczma 1980].

Sei F die Menge aller analytischen Funktionen $f: \mathbb{R} \to \mathbb{R}$. Auf ganz F ist die Iteration $f \to f^2$ möglich und $G \subset F$ sei die Menge aller iterierten Funktionen $g = f^2, f \in F$. Für alle $g \in G$ existiert somit definitionsgemäß eine iterative Wurzel, für $g \notin G$ nicht. Entsprechend kann man für höhere Ordnungen argumentieren. Allerdings ist der Nachweis, ob eine gegebene Funktion tatsächlich eine Wurzel besitzt, nicht immer trivial.

Untersuchungen über die Struktur von G haben überraschenderweise gezeigt, dass G *nirgendwo dicht* in der Menge der analytischen Funktionen ist. Das heißt, Funktionswurzeln existieren nur auf singulären Teilmengen [Blokh 1992], im Allgemeinen hat eine Funktion keine iterative Wurzeln. Allerdings trifft dieses auf eine Reihe von „relevanten“ Funktionen nicht zu.

2.3.2 Iterative Wurzeln von Polynomen

Polynome gehören zu den am besten bekannten Funktionstypen. Doch selbst in der Klasse der reellen und komplexen Polynome ist eine allgemeine Lösung für iterative Wurzeln nicht in Sicht [Baron & Jarzyk 2001]. Einige bisher gefundene Theoreme sind:

Reelle Polynome 2. Grades $f(x) = ax^2 + bx + c$ haben mindestens dann eine Funktionswurzel, wenn $a \geq 0$ oder $f(x) < x$ für alle x [Bronsthein 1989].

Die Antwort auf die Frage: „Wann ist $f(f(z)) = az^2 + bz + c$ für jedes z?“ lautet: Nie [Rice et al. 1980]! Komplexe Polynome 2. Grades besitzen keine Wurzeln auf ganz C. Dagegen existieren Wurzeln mancher quadratischen Polynome auf 1-dimensionalen Teilmengen der komplexen Ebene wie der reellen Achse oder dem Einheitskreis [Zdun 2000].

Noch völlig ungeklärt ist, ob alle komplexe Polynome 3. Grades eine Funktionswurzel besitzen [Baron & Jarzyk 2001].

Reelle Polynome 4. Grades der Form

$$f(x) = a_4x^4 + a_3x^3 + a_2x^2 + (a_1 + 1)x + a_0,\ a_4 \neq 0$$

haben genau dann ein Polynom 2. Grades als Funktionswurzel, wenn eine Kubikwurzel $\Upsilon = \sqrt[3]{a_4}$ existiert, so dass

$$4a_4A_4 = A_2^2 + 2\Upsilon A_2 \text{ und } A_3 = -1$$

mit

$$A_m = \sum_{k=0}^{m} \frac{(-1)^k}{4^k}\binom{4-m+k}{k}\left(\frac{a_3}{a_4}\right)^k a_{(4-m+k)}$$

für $m \in \{0, 1, 2, 3, 4\}$ [Schweizer & Sklar 1988].

Tschebycheff Polynome n-ten Grades haben eine Quadratwurzel genau dann, wenn $n \equiv 0$ oder $n \equiv 1 (mod\ 4)$ [Rice 1979].

Ein aktueller Überblick über iterative Wurzeln von Polynomen findet sich bei Choczewski [1992].

2.3.3 Monotone Funktionen

Iterative Wurzeln von monotonen Funktionen auf $\mathbb{R}$ gehören zu den einfachsten Problemen dieses Gebietes:

Funktionswurzeln beliebiger Ordnung $\varphi = f^{1/k}$ existieren für alle streng monoton steigenden stetigen reellwertigen Funktionen.

Von streng monoton fallenden reellwertigen Funktionen gibt es nur streng monoton fallende Wurzeln ungerader Ordnung.

Für die iterative Wurzel der Exponentialfunktion $f(x) = e^x$ fand Hardy [1924] eine geometrische Konstruktion und Kneser [1950] eine analytische Wurzel. Hardys

Methode ist allgemein anwendbar zur Konstruktion von iterativen Wurzeln monoton steigender stetiger Funktionen und soll hier kurz skizziert werden:

1. Wähle x_0 mit $f(x_0) \neq x_0$ und y_0 mit $y_0 \in [x_0, f(x_0)]$. Setzte $\varphi(x_0) = y_0$. Damit ist wegen $\varphi(\varphi(x)) = f(x)$ auch $\varphi(y_0) = f(x_0)$ festgelegt. Ebenso ist mit $x_{n+1} = y_n$ und $y_{n+1} = f(x_n)$ rekursiv eine unendliche Folge von Funktionswerten $\varphi(x_n) = f(y_n)$ bestimmt.
2. Wähle nun eine beliebige monoton steigende stetige Funktion $\varphi(x)$ auf dem Intervall $[x_0, x_1]$ mit $\varphi(x_0) = x_1$ und $\varphi(x_1) = x_2$. Damit ist nach 1. aber φ implizit auf ganz $\mathbb{R}$ festgelegt.

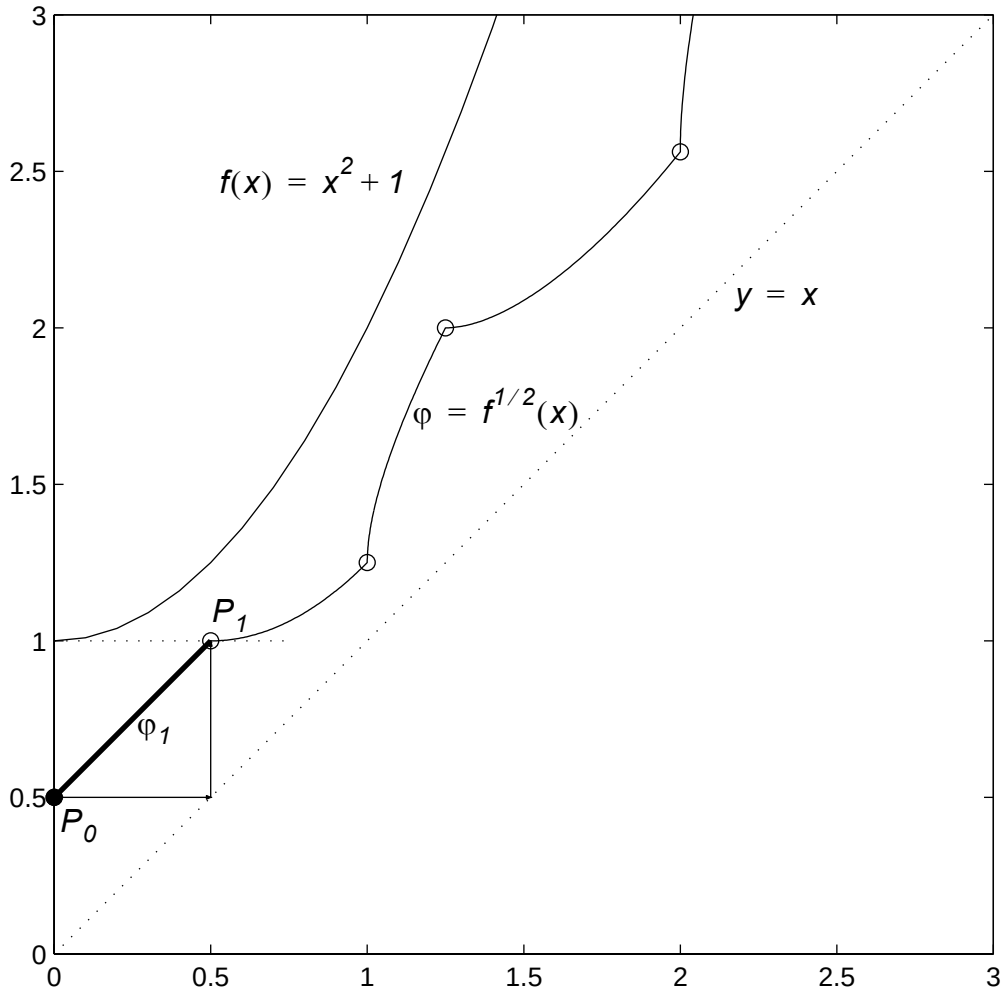

Abbildung 2.1 Konstruktion einer Funktionswurzel. Nach Hardy [1924]. Als Beispiel ist $f(x) = x^2 + 1$ dargestellt. Als Startwert wird der Punkt P_0 mit $x_0 = 0$, $y_0 = 0,5$ gewählt und damit ist die Folge der P_i festgelegt. Dann wird die Gerade φ_1 von P_0 nach P_1 als Abschnitt der Wurzel gewählt, was den Verlauf von φ auf ganz $\mathbb{R}$ festlegt.

2.4 Eindeutigkeit von Funktionswurzeln

Die eben durchgeführte Konstruktion zeigt, dass mit eindeutigen Lösungen nicht zu rechnen ist. Schon Babbage [1815] hat für die Gleichung $\varphi^2(x) = x$ neben der trivialen Lösung $\varphi(x) = x$ noch viele weitere prinzipielle Lösungen angegeben, die sogenannten „Wurzeln der Identität“, z.B:

$$\varphi(x) = a - x,\ \frac{a}{x},\ \frac{x}{ax-1},\ \frac{b-x}{1-ax},\ \frac{ax+\sqrt{b+a^2-4x^2}}{2} \quad \text{oder } g^{-1}(a-g(x))\ldots$$

Sehr oft gibt es ganze Funktionsräume von Lösungen. Oft kann man jedoch viele Lösungen durch die Forderung nach weiteren Bedingungen ausschließen wie Stetigkeit, Monotonie, positive Steigung, Differenzierbarkeit, etc.

Eine starke Bedingung ist die Forderung nach Analytizität, der Darstellbarkeit durch eine Potenzreihe. In vielen Fällen wird dadurch eine Eindeutigkeit der Lösung gewährleistet [Kuczma 1990].

2.4.1 Abhängigkeit von einer beliebigen Funktion

Wie eben gezeigt, kann eine iterative Wurzel von einer auf einem Teilintervall fast beliebig wählbaren Funktion abhängen. Dies ist ein generelles Phänomen bei Funktionalgleichungen, die gewöhnlich eine deutlich größere Lösungsmannigfaltigkeit aufweisen als beispielsweise Differentialgleichungen [Targonski 1981].

Einfachstes Beispiel für die Abhängigkeit der Lösungen einer Funktionalgleichung von einer beliebigen Funktion ist die Gleichung

$$\varphi(x) = \varphi(x+t),$$

die als Lösung *alle* periodischen Funktionen mit Periode t hat. Man definiert $\varphi(x)$ zunächst völlig beliebig auf einem halboffenen Intervall $(a, a+t]$ und erhält den Verlauf auf ganz $\mathbb{R}$ als natürliche Fortsetzung.

Selbst bei so einfachen linearen Funktionen, wie $f(x) = x + a$, können neben der trivialen Lösung noch beliebig viele oszillierende Lösungen existieren.

2.5 Wurzeln von linearen reellwertigen Funktionen

Für lineare reellwertige Funktionen f kann man die regulären Lösungen von $\varphi^n = f$ recht leicht durch einen linearen Ansatz berechnen und erhält die ebenfalls linearen iterativen Wurzeln.

2.5.1 *f*(*x*) = *ax*

$$f(x) = ax \quad \Rightarrow \quad f^{1/n}(x) = a^{1/n}x = \sqrt[n]{a}\cdot x$$

Für $a > 0$ ist dies allgemein gültig, es lässt sich sogar die reelle Iterierte $f^r(x) = a^r x$ für $r \in \mathbb{R}$ angeben. Zur Kontrolle: $f^0(x) = a^0 x = x = id$ und $f^{-1}(x) = \frac{x}{a}$ ist tatsächlich die Umkehrfunktion von f.

Ist n gerade, dann ist auch $f^{1/n}(x) = -(a^{1/n})x$ eine Lösung.

Bei negativem a ist die Funktionswurzel dagegen im allgemeinen Fall nur für ungerade n definiert.

2.5.2 f(x) = b

Eine konstante Funktion hat genau sich selbst als Iterierte und Wurzel jeder Ordnung.

$$f(x) = b \quad \Rightarrow \quad f^r = b \text{ für alle } r \neq 0$$

2.5.3 *f(x) = x + b*

$$f(x) = x + b \quad \Rightarrow \quad f^{1/n}(x) = x + \frac{b}{n}$$

und die kontinuierliche Iteration ist $f^r(x) = x + rb$ für alle r.

2.5.4 *f(x) = ax + b*

Hier ist die Lösung für beliebiges n schon deutlich schwerer zu erraten. Als linearen Ansatz probiert man $f^{1/n}(x) = \alpha x + \beta$

Für $n = 2$ erhält man damit die Gleichung

$$\alpha(\alpha x + \beta) + \beta = ax + b$$

$$\alpha^2 x + \alpha\beta + \beta = ax + b,$$

die von den Funktionen

$$f^{1/2}(x) = \pm\sqrt{a}x + \frac{b}{\pm\sqrt{a} + 1}$$

gelöst wird.

Für allgemeines $n \in \mathbb{N}$ erhält man aus dem linearen Ansatz

$$\alpha^n x + \beta \sum_{i=0}^{n-1} \alpha^i = ax + b.$$

Damit erhält man die n-te iterative Wurzel einer allgemeinen linearen Funktion $f(x) = ax + b$

$$f^{1/n}(x) = \sqrt[n]{a}x + \frac{b}{\sum_{i=0}^{n-1} a^{i/n}}.$$

Positive ganzzahlige Iterationen von $f(x) = Ax + B$ berechnen sich als

$$f^m(x) = A^m x + B \sum_{i=0}^{m-1} A^i.$$

Rationale Iterationen erhält man dann durch Einsetzen von $A = a^{1/n}$, $B = \frac{b}{\sum_{i=0}^{n-1} a^{i/n}}$:

$$f^{m/n}(x) = a^{m/n}x + b \frac{\sum_{i=0}^{} a^{i/n}}{\sum_{i=0}^{n-1} a^{i/n}}$$

$$= a^{m/n}x + b \frac{a^{m/n} - 1}{a - 1}$$

Bei negativem a gelten diese Lösungen nur für ungerades n.

2.5.5 Eindeutigkeit

Lineare Funktionen mit positiver Steigung haben genau eine lineare iterative Wurzel mit positiver Steigung. Fallende Funktionen haben nur jeweils eine ebenfalls fallende Wurzel ungerader Ordnung.

Es gibt neben diesen linearen Wurzeln linearer Funktionen allerdings noch weitere, um die linearen herum oszillierende Lösungen.

2.6 Allgemeine reellwertige Funktionen

2.6.1 Existenz

Relativ einfach ist der Fall bei stetigen, streng monoton steigenden Funktionen. Sie besitzen ebenfalls stetige, streng monoton steigende Wurzeln beliebiger Ordnung, dazu streng monoton fallende Wurzeln gerader Ordnung.

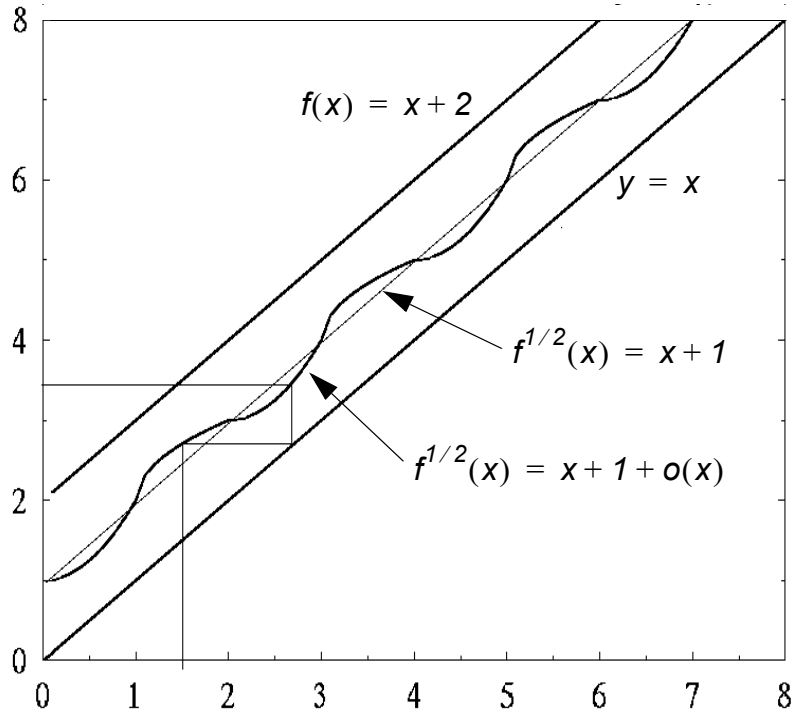

Abbildung 2.2 Eine oszillierende Wurzel von f(x)=x+2. Zusätzlich zur linearen Lösung gibt es unendlich viele, die um deren Graph f(x)=x+1 oszillieren.

Streng monoton fallende Funktionen haben streng monoton fallende Wurzeln ungerader Ordnung.

2.6.2 Invarianzen

Sei $\varphi(x)$ die iterative Wurzel von $f(x)$. Dann ist für jede bijektive Transformation $T: \mathbb{R} \to \mathbb{R}$ die Funktion $T(\varphi(T^{-1}(x)))$ eine iterative Wurzel von $T(f(T^{-1}(x)))$.

Beweis:

$$\begin{aligned} T(f(T^{-1}(x))) &= T(\varphi(T^{-1}(T(\varphi(T^{-1}(x)))))) \\ &= T(\varphi(\varphi(T^{-1}(x)))) \\ &= T(\varphi^2(T^{-1}(x))) \end{aligned}$$

Für höhere Wurzeln lasst sich dies entsprechend zeigen. Hilfreich für die spätere Berechnung mit neuronalen Netzen ist vor allem die lineare Transformation zur (gemeinsamen) Normierung von Ein- und Ausgängen:

Sei $\varphi(x)$ eine n-te Wurzel von $f(x)$. Dann ist $\frac{\varphi(ax+b)-b}{a}$ die n-te Wurzel von $\frac{f(ax+b)-b}{a}$. Das bedeutet, dass die Graphen von Funktion und Funktionswurzeln gemeinsam skalierbar und parallel zur Winkelhalbierenden verschiebbar sind.

2.7 Lineare Abbildungen - Wurzeln und Potenzen von Matrizen

Lineare Transformationen $F: \mathbb{R}^n \to \mathbb{R}^n$ ohne Offset kann man als $n \times n$ Matrix schreiben. Daraus die (Funktions)Wurzel zu ziehen, bedeutet, die Matrixgleichung $\Psi \cdot \Psi = F$ zu lösen.

Für die 2. Wurzel von 2×2 Matrizen $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ kann man dies noch explizit ausführen:

$$\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \cdot \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\begin{bmatrix} \alpha^2 + \beta\gamma & \alpha\beta + \beta\delta \\ \alpha\gamma + \gamma\delta & \beta\gamma + \delta^2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\begin{bmatrix} \alpha^2 + \beta\gamma - a & \alpha\beta + \beta\delta - b \\ \alpha\gamma + \gamma\delta - c & \beta\gamma + \delta^2 - d \end{bmatrix} = 0$$

Diese Matrixgleichung kann man in vier einfache Gleichungen mit vier Unbekannten aufteilen:

$$\begin{aligned} \alpha^2 + \beta\gamma - a &= 0 \\ \alpha\beta + \beta\delta - b &= 0 \\ \alpha\gamma + \gamma\delta - c &= 0 \\ \beta\gamma + \delta^2 - d &= 0 \end{aligned}$$

Man erhält vier Lösungen, basierend auf den Termen

$$D_{1..4} = \pm \frac{\sqrt{(a^2 + 4bc - 2ad + d^2)(\pm (a + d) + 2\sqrt{ad - bc})}}{a^2 + 4bc - 2ad + d^2}$$

mit den Matrixelementen $\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{\frac{1}{2}}$

$$\alpha = \frac{1+(a-d)D_i^{\,2}}{2D_i}$$

$$\beta = \frac{b(2(a+d)D_i^{\,2}-1)}{(a^2+4bc-2ad+d^2)D_i^{\,3}}$$

$$\gamma = cD_i$$

$$\delta = \frac{1+(d-a)D_i^{\,2}}{2D_i}$$

Für die 3. Wurzel einer 2 x 2 Matrix treten schon gekoppelte Gleichungssysteme mit Polynomen 3. Grades auf, was die Berechnung extrem aufwendig macht. Auch bei größeren Matrizen werden die Gleichungen recht unhandlich. Daher ist es nützlich, folgende allgemeine Definition und Rechenvorschrift zu verwenden.

2.7.1 Potenzen einer Matrix

Eine weitere Methode, die allgemeine Potenz einer Matrix zu definieren und zu berechnen, basiert auf Diagonalmatrizen D:

$$D = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & \ldots \end{bmatrix} \quad a, b, c, \ldots \in \mathbb{R}$$

Bei ihnen lässt sich die r-te Potenz auf natürliche Weise definieren durch

$$D^r = \begin{bmatrix} a^r & 0 & 0 & 0 \\ 0 & b^r & 0 & 0 \\ 0 & 0 & c^r & 0 \\ 0 & 0 & 0 & \ldots \end{bmatrix}, \; r \in \mathbb{R}.$$

Für ganzzahlige r entspricht das genau der r-fachen Matrixmultiplikation und es gelten die Rechenregeln für Potenzen. Damit ist dies eine sinnvolle Erweiterung der Definition.

Nicht diagonale Matrizen F kann man durch folgendes Verfahren entsprechend behandeln: Man bestimme die Eigenwerte k_i von F sowie die korrespondierenden Eigenvektoren s_i. Dann ist per Definition

$$F \cdot s_i = k_i \cdot s_i .$$

Erzeugt man die Matrix T mit den s_i als i-tem Spaltenvektor und die Diagonalmatrix D mit den k_i als Diagonalelementen, kann man das schreiben als

$$FT = TD.$$

Multipliziert man diese Gleichung noch mit T^{-1}, erhält man wegen $TT^{-1} = T^{-1}T = Id$:

$$FTT^{-1} = F = TDT^{-1}$$

Quadrieren der Gleichung ergibt

$$F^2 = TDT^{-1}TDT^{-1} = TD^2T^{-1}$$

und durch Induktion erhält man für andere Exponenten:

$$F^{r+1} = F^rF = TD^rT^{-1}TDT^{-1} = TD^{r+1}T^{-1}$$

Für ganzzahliges r gilt dann ganz allgemein

$$F^r = TD^rT^{-1}.$$

Lässt man auch hier $r \in \mathbb{R}$ zu, kann somit auch für alle quadratischen Matrizen, die positive reelle Eigenwerte besitzen, eine allgemeine Potenzfunktion bzw. eine *reelle Iteration* definiert und berechnet werden.

Standardbibliotheken zur linearen Algebra, z.B. Linpack, Eispack und Matlab, enthalten diese Methode zur Berechnung von Matrix-Potenzen. In Matlab z.B. sind die Funktionen und sqrtm(A) und mpower(A,x) vordefiniert.

2.7.2 Abgrenzung der Begriffe

Die hier verwendeten Begriffe und Definitionen können ggf. zu Verwechslungen oder Mehrdeutigkeiten mit ähnlichen Namen aus anderen Gebieten führen, deshalb die folgenden Erläuterungen:

Das Quadrat einer Matrix A^2 ist etwas anderes als ihre quadratische Form $Q_A(x) = x^TAx$.

In der Prozessidentifikation taucht gelegentlich der Begriff der „Quadratwurzel“ einer Matrix auf. Gemeint ist damit jedoch die Aufteilung einer Matrix P in zwei Dreiecksmatrizen, $P = SS^T$, die dann ebenfalls als Wurzeln von P bezeichnet werden. Ziel ist meist, dadurch die numerischen Eigenschaften von P zu verbessern, was in sogenannten Wurzelfiltern eingesetzt wird [Isermann 1992].

2.7.3 Geometrische Interpretation

Beispiel: Gegeben ist eine geometrische Transformation F, hier eine Drehung des Vektors $x \in \mathbb{R}^2$ um den Winkel ω

$$F \cdot x = \begin{bmatrix} \cos\omega & -\sin\omega \\ \sin\omega & \cos\omega \end{bmatrix} x .$$

Kann diese Abbildung in mehrere Schritte aufgeteilt werden? Die intuitive Lösung

$$F^{1/n} \cdot x = \begin{bmatrix} \cos\frac{\omega}{n} & -\sin\frac{\omega}{n} \\ \sin\frac{\omega}{n} & \cos\frac{\omega}{n} \end{bmatrix} x$$

ist tatsächlich eine Funktionswurzel von F, wie man leicht nachrechnen kann. Weitere Lösungen finden sich in Abbildung 3.8.

2.8 Berechnung von Funktionswurzeln

Es gibt kein Verfahren, das es auf einfache und allgemeine Weise erlaubt, Funktionswurzeln und fraktionale Iterationen für nichtlineare Abbildungen zu bestimmen.

Leider unterstützt auch keines der bekannten Mathematikpakete, wie Matlab, Mathematica oder Maple, die analytische oder numerische Berechnung fraktionaler Iterationen. Maple z.B. hat zwar den Operator „iterate“ bzw. „@@“ für Funktionen deklariert, allerdings nur für ganzzahlige Argumente: f^{-1} bedeutet die Inverse, f^0 die Identität und $f^{n \in \mathbb{N}}$ die n-te Iterierte der Funktion f.

Matlab besitzt immerhin die Funktion *mpower(),* mit der eine quadratische Matrix potenziert werden kann, und unterstützt auch die Exponentationschreibweise für solche Matrizen.

2.9 Bernoulli Shifts

Eine interessante Anwendung von iterativen Wurzeln sind Bernoulli Shifts, die elementaren Abbildungen der Ergodentheorie. Ergodische Abbildungen sind immer einbettbar in einen kontinuierlichen Fluss. Daher müssen auch ihre iterativen Wurzeln existieren [Ornstein 1974]. Allerdings können diese recht seltsame Eigenschaften haben. Der Bernoulli-Shift $\left(\frac{1}{2}, \frac{1}{2}\right)$ z.B., die sogenannte Baker-Abbildung, bildet das Einheitsquadrat auf sich selbst ab [Schroeder 1991]. Wie sieht eine Abbildung aus, die dies in zwei identischen Schritten erreicht?

Die Baker Abbildung ist gegeben durch $(x_{n+1}, y_{n+1}) = f(x_n, y_n)$ mit

$$x_{n+1} = (2x_n) \bmod 1$$

$$y_{n+1} = \begin{cases} y_n/2 & \text{wenn } x_n < 1/2 \\ (y_n + 1)/2 & \text{wenn } x_n \geq 1/2 \end{cases}$$

In Binärschreibweise entspricht dies dem Verschieben aller Ziffern von x um eine Stelle nach links, gefolgt vom Abschneiden des evtl. entstandenen Vorkommateiles. Dieser wird stattdessen zu y addiert, welches dann um eine Stelle nach rechts verschoben wird.

Beispielsweise wird aus $\begin{matrix} x_0 = 0,\mathbf{1}0110... \\ y_0 = 0,01110... \end{matrix}$ durch Anwenden von f $\begin{matrix} x_1 = 0,0110... \\ y_1 = 0,\mathbf{1}01110... \end{matrix}$

Fügt man beide Binärbrüche x und y zu einer einzigen Folge von Ziffern zusammen, indem man die Nachkommastellen von y in *umgekehrter* Reihenfolge vor x schreibt, erhält man eine *beidseitig* unendliche Folge von Bits:

$$\begin{matrix} 0, x^1x^2x^3\ldots \\ 0, y^1y^2y^3\ldots \end{matrix} \Rightarrow \ldots y^3y^2y^1, x^1x^2x^3\ldots \tag{2.3}$$

Die Baker-Abbildung f entspricht somit einem Takt eines unendlich langen Schieberegisters. Eine iterative Wurzel zu finden, bedeutet, eine Schaltung zu konstruieren, die für das Verschieben um eine Stelle zwei identische Taktzyklen benötigt.

Eine allgemeine Lösung dieses Problems ist nicht bekannt, jedoch kann man für y mit *endlich* vielen Stellen die in (2.3) konstruierte „Zahl“ einfach mit $\sqrt{2}$ multiplizieren. Diese Operation zweimal angewandt ist gerade die Multiplikation mit 2, also f. Allerdings ist diese iterative Wurzel der Baker-Abbildung extrem unstetig und verwischt: Das „least significant Bit“ von y beeinflusst alle anderen Stellen von x und y.

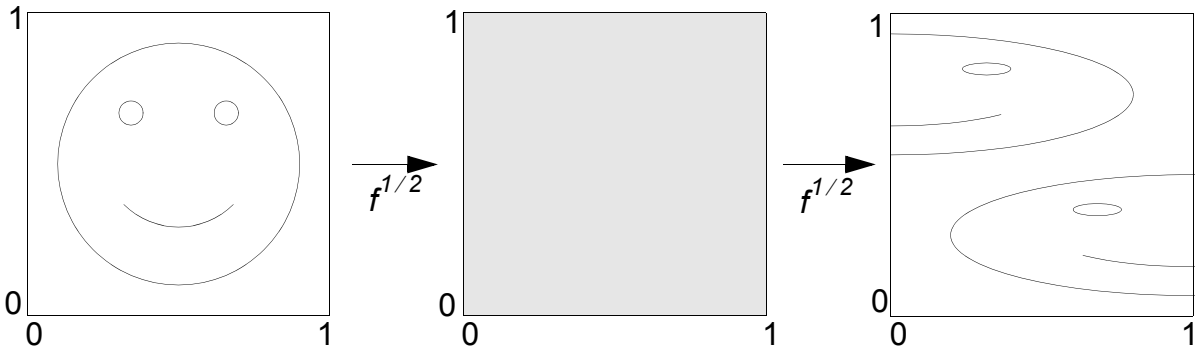

Abbildung 2.3 Eine Wurzel der Bernoulli Shift Abbildung

2.10 Automaten und ihre Wurzeln

2.10.1 Definition

Lösungen φ der Gleichung

$$\varphi(\ldots\varphi(x, p_1), \ldots, p_n) = f(x, p_1, \ldots, p_n) \tag{2.4}$$

für eine gegebene Funktion f und Parameter p_i werden im Folgenden als *parametrierte Funktionswurzeln* von f bezeichnet.

Eine Interpretation dieser Gleichung ist in der Automatentheorie zu finden: f ist die Funktion, die beschreibt, wie sich der Zustand x eines Automaten $f(x, p_1 \ldots p_n)$, der die Eingaben $p_1 \ldots p_n$ erhält, in einem Schritt ändert. Gesucht ist ein Automat $\varphi(x, p)$ mit gleichem Zustand x, der diese Eingänge stattdessen einzeln sukzessive in n Schritten erhält, danach aber den gleichen Endzustand wie f annimmt. Man könnte φ als iterative Wurzel des Automaten f bezeichnen [Targonski 1981].

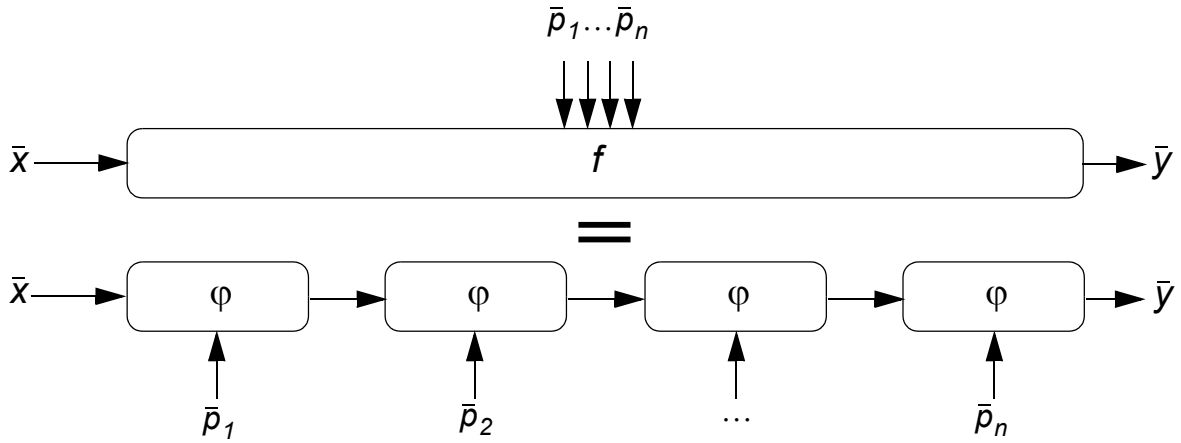

Abbildung 2.4 Schematisierung des erweiterten Funktionswurzelbegriffes. Eine Funktion f ist in eine Verknüpfung von n identischen Teilfunktionen φ zu zerlegen, so dass $y = f(x, p_1, p_2, \ldots, p_n) = \varphi(\ldots\varphi(\varphi((x, p_1), p_2))\ldots, p_n)$ gilt.

Auch eine Interpretation im Bereich der dynamischen Systeme ist einleuchtend: Die Funktion f aus Gleichung (2.4) beschreibt die Zeitentwicklung eines diskreten, *nicht* autonomen dynamischen Systems, das von außen durch die Größen p_i beeinflusst wird. Um wie bei den iterativen Wurzeln einen Übergang zur kontinuierlichen Dynamik zu vollziehen, muss allerdings auch der Einfluss p in einer kontinuierlichen Form dargestellt werden, etwa in der Art $x(t) = f^t(x_0, p(t))$.

2.10.2 Lineare Lösungen

Für diese Art der fraktionalen Iteration finden sich in der Literatur praktisch keine Hilfestellungen. Für lineare Übertragungsfunktionen kann man jedoch die linearen Wurzeln leicht berechnen.

Setzt man für $\varphi(x, p_i) = \alpha x + \beta p_i + \gamma$, ergibt sich durch n-faches Iterieren

$$\varphi^n = \alpha^n x + \sum_{i=1}^{n} \alpha^{n-i}(\beta p_i + \gamma).$$

2.11 Fraktionale Ableitungen und Integrale

Wesentlich bekannter als die nicht ganzzahlige Iteration ist die fraktionale Differential- und Integralrechnung mit Anwendungen, z.B. in Diffusionsprozessen:

Sei $f^{[0]}(x) = f(x)$ eine gegebene Funktion, dann lassen sich Ableitungen und Stammfunktionen n-ten Grades induktiv definieren als

$$f^{[n+1]}(x) = \frac{\mathrm{d}}{\mathrm{d}x} f^{[n]}(x) \quad \text{und} \quad f^{[n-1]}(x) = \int_0^x f^{[n]}(s)\,ds.$$

$f^{[n]}$, $n \in \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ bezeichnet also die n-te Ableitung von f für positives n, das n-fach ausgeführte Integral, wenn $n < 0$ ist. Dabei gilt für alle $m, n \in \mathbb{N}_0$

$$(f^{[m]})^{[n]} = (f^{[n]})^{[m]} = f^{[m+n]}.$$

Man kann nun versuchen, $f^{[t]}$ auch für nichtganzzahliges $t \in \mathbb{R}$ zu verallgemeinern: Der Ausdruck

$$f^{[-1/2]}(x) = \frac{1}{\sqrt{\pi}} \int_0^x \frac{f(s)}{\sqrt{x-s}}\,ds$$

definiert z.B. sinnvoll ein Halb-Integral von f, denn die 2-fache Anwendung

$$(f^{[-1/2]})^{[-1/2]} = f^{[-1]}$$

ergibt tatsächlich das Integral von f. Die halbe Ableitung $f^{[1/2]}$ erhält man dann durch Ableiten von $f^{[-1/2]}$:

$$f^{[1/2]} = (f^{[-1/2]})^{[1]}.$$

Die allgemeine Form dieses Liouville-Riemann-Weyl Integrals lautet [Spanier 1974]:

$$f^{[-v]}(x) = \frac{1}{\Gamma(v)} \int_0^x (t-s)^{v-1} f(s)\, ds$$

Besonders einfach ist es, bei trigonometrischen Funktionen eine verallgemeinerte Ableitung zu definieren. Ableiten bedeutet hier eine Phasenverschiebung um $\pi/2$, n-faches Ableiten um $n\pi/2$. Entsprechend wählt man formal einfach eine Phasenverschiebung von $t\pi/2$, $t \in \mathbb{R}$ für die verallgemeinerte Ableitung oder Integration.

Ähnlich wie die fraktionale Iteration ist auch die fraktionale Differentialrechnung bei Mathematikern seit langem bekannt. Praktische Anwendungen in Physik und Ökonomie werden aber auch erst seit wenigen Jahren betrieben, die Mehrzahl aller Arbeiten stammen aus der Zeit seit 1995.

Ob zwischen der fraktionalen Iteration einer Funktion und ihrer fraktionalen Ableitung eine Beziehung besteht, wurde bisher nicht untersucht. Die Literatur beider Gebiete ist disjunkt.

Dabei gibt es einen fundamentalen Zusammenhang auf der Ebene des Differentialoperators selbst: Der fraktionale Differentialoperator ist gerade die fraktionale Iteration des normalen Differentialoperators, denn n-faches Ableiten bedeutet n-maliges Anwenden des Operators

$$\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^n = \frac{\mathrm{d}^n}{\mathrm{d}x^n}.$$

Entsprechend ist auch die Einbettung $n \in Z \Rightarrow r \in \mathbb{R}$ gegeben.

Es wäre interessant zu betrachten, ob einige der Theoreme der Iterationstheorie bei verallgemeinerten Ableitungen und Integralen hilfreich sein könnten. Der Splinter einer Funktion f ist dann die unendliche Folge ihrer n-ten Ableitungen. Ein Orbit des Differentialoperators ist die Menge aller Funktionen, die irgendwann von diesem Operator auf eine gemeinsame Funktion abgebildet werden.

Beispielsweise gehören alle Polynome unter der Iteration des Ableitungsoperators zum gleichen Orbit, der einen Zyklus der Länge 1 hat: $f(x) = 0$.

$sin(x) + c$ und $cos(x) + d$ für alle c, d sind Elemente eines Orbits, der einen Zyklus der Länge 4 besitzt: $sin(x) \Rightarrow cos(x) \Rightarrow -sin(x) \Rightarrow -cos(x) \Rightarrow sin(x)$.

KAPITEL 3

Lösung mit Neuronalen Netzen

Das Problem, iterative Wurzeln und fraktionale Iterationen einer Funktion zu bestimmen, kann mit neuronalen Netzen, die normalerweise zur Approximation von Funktionen verwendet werden, angegangen werden. Dabei wird von der Fähigkeit eines Netzes ausgegangen eine beliebige Funktion zu modellieren und dazu eine Topologie verwendet, die in Verbindung mit darauf angepassten Lernverfahren eine Funktion als iterativen Prozess modelliert, dessen Teilschritt dann einer iterativen Wurzel entspricht.

3.1 Multilayer Perzeptrons

Mehrschichtige Perzeptrons (Multilayer Perzeptrons) kurz MLPs sind eine der Standardarchitekturen für Neuronale Netze. MLPs sind in Schichten strukturiert, die aus einzelnen Neuronen bestehen.

3.1.1 Neuronenmodell

Jedes Neuron bildet n Eingänge x_i durch gewichtete Summation und eine Aktivierungsfunktion ϕ auf einen Ausgangswert y ab:

$$y = \phi\left(b + \sum_i^n w_i x_i\right)$$

Die Werte w_i werden als die Gewichte, b als das Bias des Neurons bezeichnet. Als Aktivierungsfunktion wird in der Regel eine sigmoide Funktion verwendet, wie z.B.

$$\phi(s) = \frac{1}{1 + e^{-s}}.$$

Oft werden speziell in Ausgabeneuronen auch lineare Aktivierungsfunktionen verwendet, das heißt, eigentlich gar keine: $\phi(s) = s$

3.1.2 Topologie

Im allgemeinsten Fall können solche Neuronen beliebig miteinander verschaltet werden, indem an den Ausgang eines Neurons die Eingänge weiterer Neuronen geschaltet werden. Jedes Neuron kann seine eigene Aktivierungsfunktion haben. In der Regel werden aber Netze mit geschichteten, nur vorwärtsprojizierenden Topologien verwendet. Solch ein MLP besteht üblicherweise aus einer Eingangsschicht, einer oder mehreren „verdeckten“ Schichten und einer Ausgangsschicht. Verknüpfungen laufen nur in einer Richtung von den Ausgängen einer Schicht zu den Eingängen der nächsten und alle Neuronen einer Schicht besitzen die gleiche Aktivierungsfunktion.

Neuronen der Eingangsschicht führen keine eigenständige Verarbeitung durch, sie dienen im Grunde nur als Abstraktion für die Eingangswerte, die von den Neuronen der verdeckten Schicht erstmals verarbeitet werden.

Netze mit ausschießlich linearen Neuronen, Adalines (Adaptive linear Networks), brauchen keine verdeckte Schicht. Sie können lediglich lineare Transformationen der Eingangsvektoren durchführen.

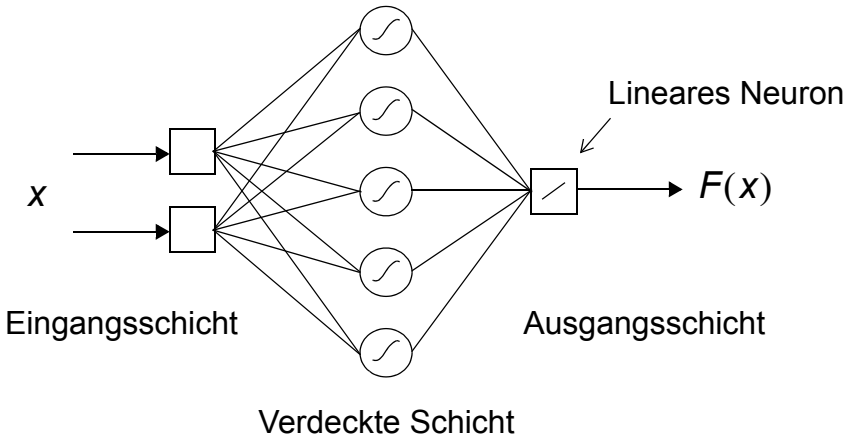

Abbildung 3.1 Ein einfaches Multilayer Perzeptron (MLP). Neuronen, an denen die Werte der Aktivierung auch nach außen Bedeutung haben, werden durch Quadrate symbolisiert, Neuronen, die nur internen Berechnungen dienen, durch Kreise. Diese Struktur kann formal als (2-5-1) abgekürzt werden

3.1.3 Lernfähigkeit

Soweit stellt ein neuronales Netz nichts weiter als eine Transformationsfunktion dar, die Eingangsvektoren auf Ausgänge abbildet. Normalerweise ist jedoch diese Abbildung nicht a priori bekannt, sondern soll erst aus Beispielen, den Trainingsdaten, erlernt werden. Diese bestehen aus einem Datensatz von Eingangswerten x (Inputs)

und dazugehörigen Zielwerten t (Targets). Als Lernen wird der Vorgang bezeichnet, die Gewichte und Biase des Netzes so anzupassen, dass das Netz den Zielwerten möglichst ähnliche Ausgangswerte y liefert.

Dazu wird eine Fehlerfunktion E definiert, die die Abweichung von Ziel- und Ausgabewerten quantifiziert, meist in Form der Summe der quadratischen Abweichungen $E = \sum (t-y)^2$. Durch ein Gradientenabstiegsverfahren auf den Gewichten des Netzes wird dieser Netzfehler schrittweise minimiert (den Faktor η bezeichnet man als Lernrate):

$$\Delta w_i = \eta \frac{\mathrm{d}E}{\mathrm{d}w_i} \tag{3.1}$$

Rummelhard et al. [1986] stellten ein sehr effizientes Verfahren vor, den Backpropagation-Algorithmus, der die Ableitungen $\frac{\mathrm{d}E}{\mathrm{d}w_i}$ rekursiv rückwärts durch das Netz schreitend berechnet, was eine drastische Einsparung an Rechenoperationen darstellt gegenüber einer jeweils eigenen Berechnung pro Gewicht.

3.2 Universelle Funktionsapproximation

MLPs werden gerne als „universelle“ Funktionsapproximatoren bezeichnet. Für ganz beliebige Funktionen gilt dies jedoch nur im folgenden, eingeschränkten Sinn:

Gegeben sei eine beliebige Funktion $y = f(x)$, $x \in \mathbb{R}^m$ $y \in \mathbb{R}$. Dann gibt es stets ein mehrschichtiges Perzeptron F mit m Eingängen, einem Ausgang und mindestens einer verdeckten Schicht, das in der Lage ist, *f auf einer endlichen* Teilmenge $T \subset \mathbb{R}^m$ von n Punkten mit beliebiger Genauigkeit zu reproduzieren. Das heißt, es gibt eine Zahl $N \in \mathbb{N}$, die minimale Anzahl der „versteckten Neuronen“ sowie einen Gewichtssatz w_i, so dass für alle $x \in T$

$$F(x) = f(x). \tag{3.2}$$

Weitergehende Forderungen, wie punktweise oder gleichmäßige Konvergenz auf einer kompakten Teilmenge $U \subset \mathbb{R}^m$, sind dagegen nicht so einfach für beliebige Zielfunktionen f zu realisieren. Die Behauptung, dass ein Netz F mit endlicher Anzahl von Neuronen in den verdeckten Schichten existiert, so dass

$$max|F(x) - f(x)| < \varepsilon \tag{3.3}$$

für alle $x \in U$ und ein beliebiges $\varepsilon > 0$, lässt sich nur für „hinreichend gutmütige“, das heißt stetige differenzierbare $f \in C(\mathbb{R}^n)$, allgemein zeigen.

Voraussetzung ist, dass die durch die gewichtete Summation der Aktivierungsfunktion erzeugbaren Netzfunktionen „dicht“ im Raum der zu approximierenden Funktionen sind. Dass es solche Funktionen geben muss, hat Kolmogorov bewiesen [Pinkus

1999]. Hornik, Stinchcombe und White [1989] zeigten, dass sigmoide Funktionen diese Bedingung erfüllen können und ein MLP mit einer verdeckten Schicht deshalb zum universellen Approximator nach Gleichung (3.3) machen.

Auf Abbildungen $\mathbb{R}^m \to \mathbb{R}^n$ lässt sich diese Argumentation einfach durch die Annahme von n parallelen Netzen übertragen.

Hauptsächlich werden neuronale Netze aber dann angewendet, wenn $f(x)$ gar nicht bekannt ist, sondern lediglich ein endlicher Datensatz $(x, y)\ y \approx f(x)$ gegeben ist, der zudem auf vielfältigste Weise rauschbehaftet, fehlerhaft, widersprüchlich oder unvollständig sein kann. Ziel ist es dann, ein Netz F als Ersatz für f zu konstruieren, das nicht nur einem minimalen Trainingsfehler $\|F(T) - f(T)\|$ auf der Trainingsmenge T, sondern möglichst auf einem ganzen Intervall $U \supset T$ aufweist. Dieses Verhalten bezeichnet man als Generalisierungsfähigkeit eines Netzes, den mittleren Fehler auf einer Stichprobe $V \subset U$, $V \cap T = \varnothing$ als den Generalisierungsfehler. Bei rauschbehafteten Trainingsdaten kann es besser sein, den Netzfehler auf der Trainingsmenge nicht gleich Null werden zu lassen, entweder durch die Wahl eines „zu kleinen“ N oder durch geeignete Trainingsverfahren (Crossvalidierungstraining).

Aufgrund von Skalierungseigenschaften und Sättigungseffekten sind MLPs nicht in der Lage, beliebige Funktionen auf *unbeschränkten* Intervallen zu approximieren. So kann selbst die „einfachste“ Funktion $f(x) = x$ von einem MLP, das in einer Schicht nur sigmoide Aktivierungsfunktionen hat, nicht auf ganz $\mathbb{R}$ angenähert werden: Die Sigmoide ist beschränkt und kann durch nachgeschaltete Multiplikation und Addition mit Gewichten und Biasen nur auf ein zwar beliebiges, aber festes Intervall abgebildet werden.

Als den Arbeitsbereich oder Vertrauensbereich eines Netzes F bezeichnet man die Menge, auf der das Netz die Funktion f mit der gewünschten Genauigkeit approximieren kann. Die Bestimmung dieses Vertrauensbereiches für einen gegebenen Trainingsdatensatz kann allerdings ähnlich kompliziert sein, wie die Funktionsapproximation selbst [Kindermann 1999]. Gebräuchlich sind daher einfache Abschätzungen, wie z.B. die konvexe Hülle von T oder der minimale achsenparallele Hyperquader, der T ganz enthält. Innerhalb dieses Bereiches spricht man von *Interpolation*, außerhalb von einer eher wenig vertrauenswürdigen *Extrapolation*.

Hat man umgekehrt ein Intervall U gegeben, auf dem eine bekannte Funktion f zu modellieren ist, wird man die Trainingsmenge $T \subset U$ so wählen, dass U vollständig abgedeckt ist und ggf. Bereiche mit starker Variation von f durch entsprechend mehr Trainingsdaten abdecken.

Bei der später behandelten Verknüpfung von mehreren Netzen kommt diesem Problem eine spezielle Rolle zu: Dort ist für jedes Teilnetz der Vertrauensbereich relevant.

3.3 Iterative Wurzel und fraktionale Iterationen

Aufbauend auf ihrer Fähigkeit zur Funktionsapproximation kann man neuronale Netze auf intuitive Weise auch dazu verwenden, iterative Wurzeln und fraktionale Iterationen zu modellieren. Lediglich einige Erweiterungen am Lernalgorithmus sind dazu notwendig.

3.3.1 Funktionsverkettung und Iteration

Die Operation der Verkettung von Funktionen kann direkt auf eine Netztopologie von MLPs abgebildet werden, in der hintereinandergeschaltete Gruppen von Schichten jeweils eine Funktionskomponente repräsentieren.

Seien die Funktionen $f(x)$ und $g(x)$, $\mathbb{R}^n \to \mathbb{R}^n$ durch MLPs repräsentiert. Dann ist die Verkettung $h(x) = g(f(x))$ einfach durch das Hintereinanderschalten der beiden MLPs gegeben. Dieses neue Netz ist ebenfalls ein MLP, bestehend aus zwei *Subnetzen*. Zu beachten sind allerdings die Wertebereiche: Um im Arbeitsbereich des g-Netzes zu bleiben, muss das Bild von x unter f im Trainingsbereich von g liegen. Andernfalls ist das Ergebnis als Extrapolation nicht vertrauenswürdig.

Hängt man zwei *identische* Netze, die eine Funktion $f(x)$ modellieren, hintereinander, so entspricht das der Iteration $f(f(x)) = f^2(x)$. Die n-te Iterierte $f^n(x)$ modelliert man entsprechend durch das Hintereinanderschalten von n identischen Netzen.

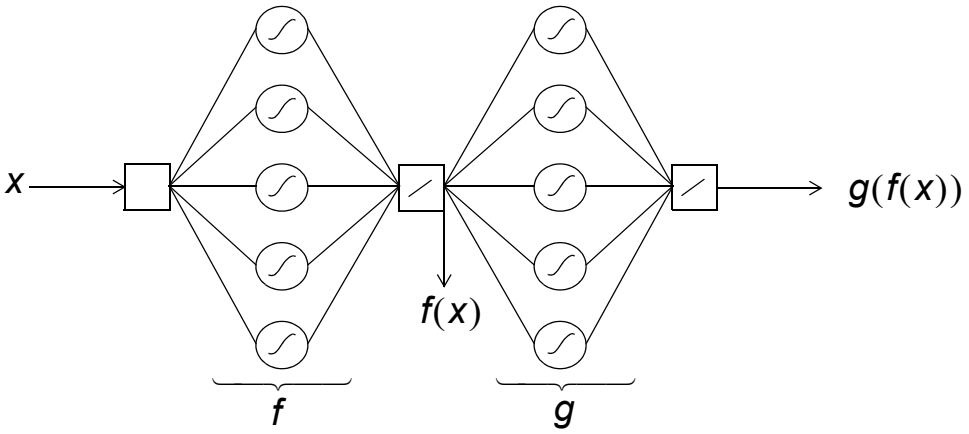

Abbildung 3.2 Funktionskomposition durch Hintereinanderschalten von Teilnetzen. Struktur: (1-5-1-5-1) oder $(1\text{-}5\text{-}1)^2$. Die Netzschichten, die Werte nach außen liefern, werden im Text als „Datenschichten“ bezeichnet.

Alternativ könnte man nur ein Netz verwenden und die Ausgabe n-fach wiederholt durch das Netz schicken.

3.3.2 Modellierung der iterativen Wurzel

Die Berechnung von iterativen Wurzeln ist gewissermaßen die Umkehrung dieser Hintereinanderschaltung von Netzen: Kann ein Netz, das die Funktion f: $\mathbb{R}^n \to \mathbb{R}^n$

approximiert, in zwei identische Teilnetze zerlegt werden, von denen jedes gerade die iterative Wurzel φ von f modelliert?

Dazu wird von vornherein eine Topologie des Netzes gewählt, die sich später leicht aufteilen lässt: Kann f durch ein Netz der Struktur k-H-k zufriedenstellend modelliert werden, sollte auch ein Netz k-H-k-H-k dazu in der Lage sein. Das erste Teilnetz braucht z.B. nur die Identität zu modellieren. k bezeichnet die Anzahl der Neuronen in der Ein- und Ausgabeschicht, H die Struktur der verdeckten Schicht oder Schichten.

Der schwierige Teil der Aufgabe besteht darin, beide Teilnetze *identisch* zu gestalten und gleichzeitig die Modellierung von f aufrechtzuerhalten.

Dies wird in vielen Fällen prinzipiell nicht möglich sein, da viele Funktionen keine iterativen Wurzeln besitzen (vgl. Kapitel 1), also nicht aufteilbar sind. Aber wenn $\varphi^2 = f$ Lösungen hat, sollte eine solche Aufteilung möglich sein, vorausgesetzt, dass sowohl f als auch φ zur Klasse der durch ein MLP darstellbaren Funktionen gehören.

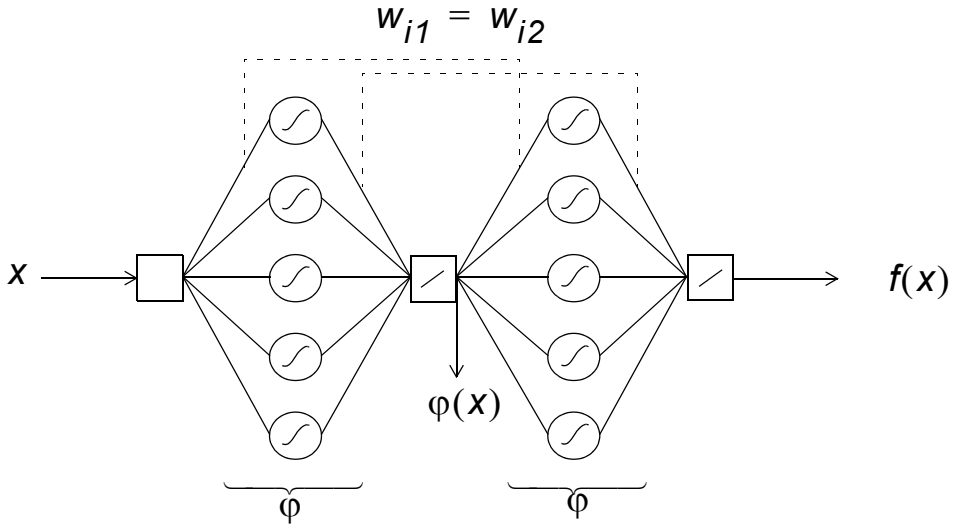

Abbildung 3.3 Modellierung der iterativen Wurzel einer Funktion *f*. Die Aufgabe ist gelöst, wenn alle korrespondierenden Gewichte beider Teilnetze übereinstimmen.

Dazu werden im Folgenden Verfahren entwickelt, die das Netztraining mit Backpropagation so erweitern, dass gleichzeitig die Angleichung der Teilnetze erfolgt.

3.3.3 Höhere Wurzeln und fraktionale Iteration

Zur Übertragung auf die n-te iterative Wurzel werden einfach n identische Teilnetze hintereinandergeschaltet, jedes davon ist dann ein Modell für $f^{1/n}$. Die fraktionale Ite-

ration $f^{m/n}$ erhält man durch anschließendes m-faches Kombinieren dieses Teilnetzes bzw. durch Auslesen des Ergebnisses nach dem m-ten Teilnetz.

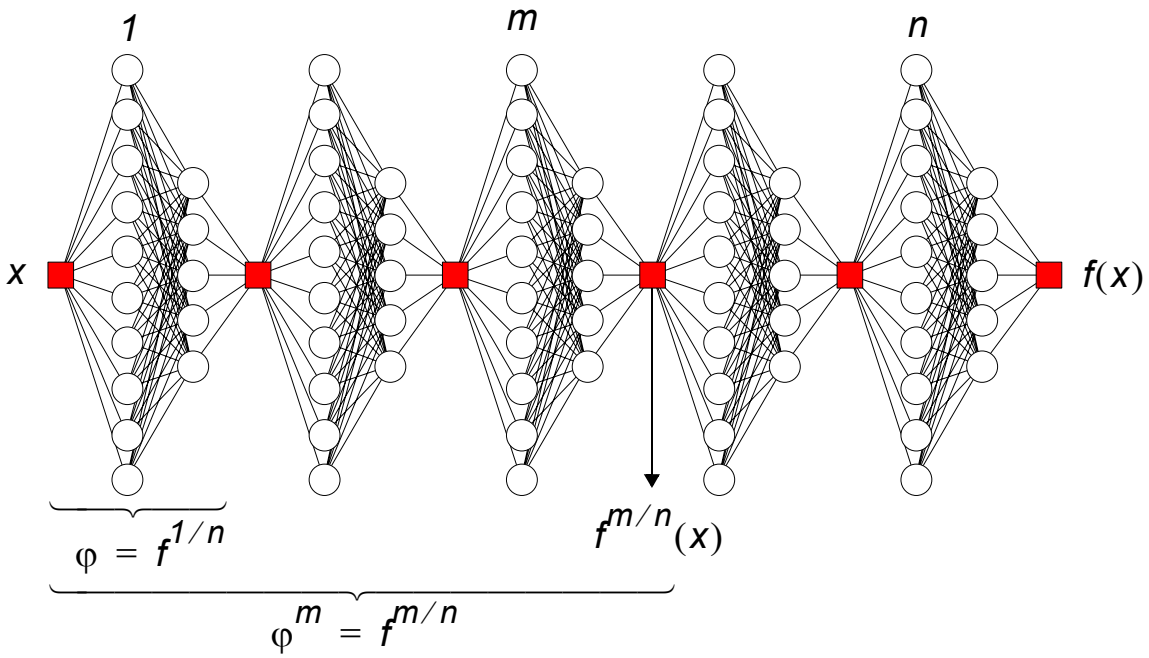

Abbildung 3.4 Netz zur Berechnung einer fraktionalen Iteration. Struktur: (1-10-5-1)^5. Bedingung für die korrekte Modellierung ist, dass alle Teilnetze identisch sind und das Gesamtnetz die Zielfunktion f approximiert.

3.3.4 Die Inverse einer Funktion

Um auch negative Iterationsindizes der verallgemeinerten Iteration zu realisieren, muss - falls existent - die Inverse von f^r berechnet werden. Zur Invertierung der Abbildungsfunktion eines neuronalen Netzes stehen eine Reihe von Verfahren zur Verfügung, im einfachsten Fall die Vertauschung von Eingangs- und Ausgangswerten beim Training. Aber auch fertig trainierte Netze können durch eine Transformation der Gewichte zur Modellierung der Inversen verwendet werden [Kindermann & Linden 1990] [Davies et.al. 1993].

3.4 Lernmethoden

Das Optimierungsproblem, die Gewichte eines Netzes so zu wählen, dass der Netzfehler minimal wird, hat in der Regel keine eindeutige Lösung. Initialisierungen mit unterschiedlichen Startwerten werden zu unterschiedlichen Gewichtsmatrizen führen, bei jeweils identischem Übertragungsverhalten des Gesamtnetzes. Die speziellen Werte einzelner Gewichte sind daher normalerweise nicht von Interesse [Chen et al. 1993]. Hat ein Netz genügend Freiheitsgrade, kann trotz Festlegung des Wertes von einzelnen Gewichten durchaus die Approximationsfähigkeit erhalten bleiben [Kröner & Moratz 1996].

Bei einem Netz, das aus zwei hintereinandergeschalteten identischen Teilen besteht, werden allerdings nicht einzelne Gewichte festgelegt, sondern jeweils zwei entsprechenden Gewichte aus beiden Netzteilen müssen den gleichen Wert haben. Bei n-ten Wurzeln gilt dies für jeweils n Gewichte.

Der Lernvorgang muss diese Vorgabe neben der Minimierung des Approximationsfehlers von f Fehlers berücksichten. Dafür sind verschiedene Verfahren geeignet.

3.4.1 Lernen nur in der letzten Schicht

Die einfachste Methode, eine solche Funktionswurzelberechnung zu implementieren, besteht darin, nur das letzte Teilnetz zu trainieren und dessen Gewichtssatz parallel dazu ständig auf die vorderen Teilnetze zu kopieren.

Man kann dies auch mit nur einem einzelnen Netz implementieren: Um die n-te Wurzel zu bestimmen, iteriert man zuerst den Input $n-1$ mal durch das Netz und benutzt das Resultat dann als Input und den gewünschten Output als Target für eine einzelne Lernepoche. Führt dieser Lernvorgang zum Erfolg, approximiert das Netz die n-te Wurzel von f.

In Anhang A ist dargestellt, wie dieses Verfahren im Neurosimulator ECANSE auf einfache Weise implementiert werden kann. ECANSE wurde als professionelles System besonders für den Bereich der Netzlastprognose entwickelt.

3.4.2 Koppeln der Gewichte

Eine „sanftere“ Methode zur Angleichung der Gewichte in den Teilnetzen ist durch das „Koppeln“ entsprechender Gewichte konstruierbar. Dieses „Weight Sharing“ wird bei neuronalen Netzen bisher schon aus verschiedenen Gründen verwendet, u.a. als Methode zur Vereinfachung von Netzen [Nolan & Hinton 1992], um z. B. bei unterdeterminierten Problemen ein Netzwerk mit weniger Freiheitsgraden zu erhalten [Lautrup et al. 1994] [Zhang & Hu 1996] oder zur Modellierung von Invarianzen [Shawe-Taylor 1994].

Um diese Methode zur Modellierung von iterativen Wurzeln zu verwenden, wird zusätzlich zur Änderung der Netzgewichte nach Gleichung (3.1), die durch das Lernen mit Backpropagation bestimmt wird, ein weiterer Term in die Gewichtsänderung eingefügt (w_{ik} bezeichnet jeweils das i-te Gewicht im k-ten Teilnetz):

$$\Delta_{Kop} w_{ia} = \beta \frac{1}{n} \sum_{k=1}^{n} (w_{ik} - w_{ia}) \tag{3.4}$$

Nach jedem Backprop Lernschritt wird dieser Term zum Gewicht addiert:

$$w_{ik}^{neu} = w_{ik}^{alt} + \Delta w_{ik} + \Delta_{Kop} w_{ik}$$

Dabei bedeutet $\beta = 0$ den normalen Backpropagation Algorithmus ohne Angleichung der Teilnetze. Mit $\beta = 1$ werden korrespondierende Gewichte nach jedem Schritt identisch.

Bei höheren Iterationstiefen tritt ein Problem beim Initiallernen auf. Das Netz braucht sehr lange, um sich überhaupt in Richtung zur Trainingsfunktion zu entwickeln. In der Regel ist die Übertragungsfunktion eines Netzes nach Initialisierung der Gewichte mit kleinen Zufallszahlen eine konstante Funktion. Bei sehr „langgestreckten“ Netzen, dazu mit Schichten, die nur aus einem Neuron bestehen, ist die Aktivierung in den hinteren Schichten praktisch nicht mehr vom Eingang abhängig - das Gradientenabstiegsverfahren bleibt lange auf einem Plateau stecken.

Ein Ausweg ist das später beschriebene Initialisieren des Netzes mit solchen Werten, dass die Übertragungsfunktion einer Schicht ungefähr die Identitätsfunktion ergibt.

Eine weiteres Verfahren, das dieses Problem reduziert, besteht darin, direkte Querverbindungen von einer Datenschicht zur nächsten zu schalten, die mit Gewichten ungleich Null initialisiert werden. Damit ist von Anfang an eine Dynamik in den Netzausgängen vorhanden.

Auch sehr hilfreich beim Lernen von iterativen Wurzeln ist das Starten mit einem nichtgekoppelten Netz und dem langsamen Anheben der Kopplungsstärke. Dies schränkt die Freiheitsgrade des Netzes nur langsam ein und vermeidet dadurch das Hängenbleiben in lokalen Minima ähnlich wie auf „simulierted Annealing” basierende Verfahren.

Diese Methode wurde als neue Option in FAST, der **F**ORWISS **A**rtificial Neural Network **S**imulation **T**oolbox implementiert, das ein in Forschung und Lehre weitverbreitetes Programm und im Internet frei verfügbar ist. Anhang C zeigt ein entsprechendes Beispiel.

3.4.3 Modifizierung der Fehlerfunktion

Eine weitere Methode, die Gewichte der einzelnen Teilnetze einander anzunähern, besteht darin, die Fehlerfunktion des Netzes dahingehend zu erweitern, dass nicht nur der Approximationsfehler auf den Trainingsdaten E_{app}, sondern auch die Abweichung der Gewichte zwischen den iLayers als zu minimierender Fehler E_{reg} berücksichtigt wird.

Gebräuchliche Fehlermaße für die Approximation bei Backpropagation Netzen sind z.B. die Summe der quadratischen Fehler, sum square error (sse), der mittlere quadratische Fehler, mean square error (mse) oder die Wurzel daraus, root mean square error (rms).

Zu diesem Fehler wird ein Term hinzugefügt, der auf irgendeine Art die Abweichung von korrespondierenden Gewichten zwischen verschiedenen iLayers bemisst:

$$E = E_{app} + E_{reg} \tag{3.5}$$

Dieser *Regularisierungsfehler* E_{reg} kann ebenfalls auf verschiedene Weise gebildet werden, „passend“ zum Approximationsfehler:

w_{ik} sei das i-te Gewicht im k-ten iLayer. Ziel ist, dass alle korrespondierenden Gewichte, also solche mit gleichem Index i, möglichst identisch werden, d.h. $w_{ij} - w_{ik}$ für beliebige j, k gegen Null strebt. Man beachte, dass i über alle Gewichte *und* Biase eines ganzen iLayers läuft, das sich ja über mehrere klassische Netzschichten erstreckt.

Ein entsprechendes Fehlermaß ist z.B. die Summe aller dieser quadrierten Gewichtsabweichungen (sum squared error, sse) mit m=Anzahl der Gewichte pro iLayer und n=Anzahl der iLayers:

$$E_{regsum} = \sum_{i=1}^{m} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} (w_{ij} - w_{ik})^2 \tag{3.6}$$

Es gibt $\frac{n(n-1)}{2}$ solcher entsprechenden *mittleren* quadratischen Abweichungen (mean squared error, mse) zweier korrespondierender Gewichte voneinander:

$$E_{regmean} = \frac{E_{regsum}}{m\frac{n(n-1)}{2}} \tag{3.7}$$

Weniger aufwendig ist die Berechnung der Summe der Varianzen der korrespondierenden Gewichte, da die Schleifenzahl mit mn statt mit mn^2 wächst. Hierzu braucht man zunächst den Mittelwert des i-ten Gewichtes über alle iLayer

$$\overline{w_i} = \frac{1}{n} \sum_{j=1}^{n} w_{i,j},$$

woraus man dann die Varianz s_i^2 erhält:

$$s_i^2 = \frac{1}{n} \sum_{j=1}^{n} (w_{i,j} - \overline{w_i})^2$$

Die Normierung erfolgt mit $1/n$, da hier kein Schätzer für die Varianz, sondern die mittlere quadratische Abweichung vom Mittelwert gesucht ist. Über alle Gewichte gemittelt ergibt sich somit eine mittlere Varianz pro Gewicht

$$E_{regvar} = \frac{1}{m}\sum_{i=1}^{m} s_i^2 . \tag{3.8}$$

Um einen Steuerungsparameter beim Training zu erhalten, kann man Gleichung (3.5) wieder um eine Wichtung $0 < \beta < 1$ erweitern:

$$E = \beta E_{app} + (1-\beta) E_{reg} \tag{3.9}$$

Kommt es vor allem auf eine genaue Anpassung der Zielfunktion an, setzt man $\beta \to 1$. Steht hingegen die Regularisierung in identische Teilnetze im Vordergrund, wählt man ein kleines β nahe Null. Dies ist vor allem interessant, wenn es keine exakte Lösung für das Funktionswurzelproblem gibt, d.h. prinzipiell nicht beide Fehleranteile gleichzeitig auf Null optimiert werden können.

Gradientenmethoden zur Fehlerminimierung benötigen die partiellen Ableitungen des Fehlers nach jedem Gewicht. Da der Regularisierungsterm zum Approximationsfehler lediglich dazuaddiert wird, ist auch die Ableitung des Gesamtfehlers eine Summe:

$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial E_{app}}{\partial w_{ik}} + \frac{\partial E_{reg}}{\partial w_{ik}} ,$$

bzw. auf Gleichung (3.9) aufbauend mit Gewichtung:

$$\frac{\partial E}{\partial w_{ij}} = \beta \frac{\partial E_{app}}{\partial w_{ik}} + (1-\beta) \frac{\partial E_{reg}}{\partial w_{ik}}$$

Die Ableitung des Approximationsfehlers wird unverändert wie üblich durch Backpropagation bestimmt. Die Ableitung des Regularisierungsterms ergibt sich für Gleichung (3.6), die Summe aller quadrierten Differenzen (sse),

$$\frac{\partial E_{regsum}}{\partial w_{ik}} = 2\sum_{j=1}^{n} (w_{ik} - w_{ij}) .$$

Für den mittleren quadratischen Fehler pro Gewicht (mse) aus Gleichung (3.7) lautet dann die partielle Ableitung entsprechend

$$\frac{\partial E_{regmean}}{\partial w_{ik}} = \frac{4}{mn(n-1)} \sum_{j=1}^{n} (w_{ik} - w_{ij})$$

Dieses Fehlermaß wird im Folgenden für das Netztraining der regularisierten Netze verwendet.

3.4.4 Wahl der Kopplungsstärke

Funktionswurzeln beliebiger Ordnung $f = F^{1/k}$ existieren definitiv für alle streng monoton steigenden reellwertigen Funktionen. In anderen Fällen gibt es oft keine Lösung. Dann kann man je nach Bedarf zwischen verschiedenen Strategien zur Wahl von Kopplungsfaktor β und Lernrate wählen:

$\beta \ll 1$ und große Lernrate => $g(f(x)) \equiv F(x)$ mit $||f, g|| \rightarrow min$
=> Beste Modellierung von $F(x)$, “Wurzeln” so ähnlich wie möglich

$\beta \cong 1$ und kleine Lernrate => $f(f(x)) \equiv G(x)$ mit $||F, G|| \rightarrow min$
=> Approximation von $F(x)$ so gut wie möglich durch Iteration von $f(x)$

3.4.5 Kontinuierliches Anheben der Kopplungsstärke

Als wirkungsvolles Verfahren hat sich herausgestellt, am Anfang des Lernvorganges mit einem kleinen Wert für die Kopplungsstärke zu beginnen, z.B. $\beta = 0,01$ und diesen dann im Laufe des Trainings langsam bis auf Eins anzuheben.

3.5 Iteration eines einzelnen Netzes

Bisher wurde von einem Netz ausgegangen, das aus hintereinandergeschalteten Teilnetzen bestand, die durch verschiedene Trainingsverfahren einander angeglichen wurden. Daher konnten bekannte Methoden, vor allem Backpropagation mit relativ einfachen Modifikationen verwendet werden.

Für hohe Iterationstiefen werden die Netze jedoch recht schwer trainierbar. Ab etwa fünf Iterationsschichten nimmt die Lernfähigkeit stark ab und die dargestellten Methoden sind nicht mehr sinnvoll einsetzbar. Je mehr iterative Schichten miteinander gekoppelt werden müssen, desto stärker ist die Einschränkung des möglichen Gewichtsraumes und es wird entsprechend schwieriger, ein Minimum zu finden.

Daher soll hier dargestellt werden, wie auch mit einem einzelnen Teilnetz, das n-fach durchlaufen wird, die iterative Wurzel einer Funktion f bestimmt werden kann. $f(x)$ soll direkt durch die Iteration eines Netzes $G(x)$ modelliert werden.

Dazu wird zunächst der Ausdruck für die Iteration eines Netzes G bestimmt, das neben der Eingangsschicht aus einer verdeckten Schicht von H nichtlinearen Neuronen mit Aktivierungsfunktion ϕ und einer linearen Ausgangsschicht besteht:

$$G(x) = \sum_{h=1}^{H} c_h \phi(a_h x + b_h)$$

Der Netzausgang nach n Iterationen $x_n = G^n(x)$ ist rekursiv gegeben durch $x_0 = x$ und

$$x_{i+1} = \sum_{h=1}^{H} c_h \phi(a_h x_i + b_h) .$$

a_h und b_h sind entsprechend Abbildung 3.5 die Gewichte und Biase der verdeckten Schicht, c_h die Gewichte der Ausgangsschicht.

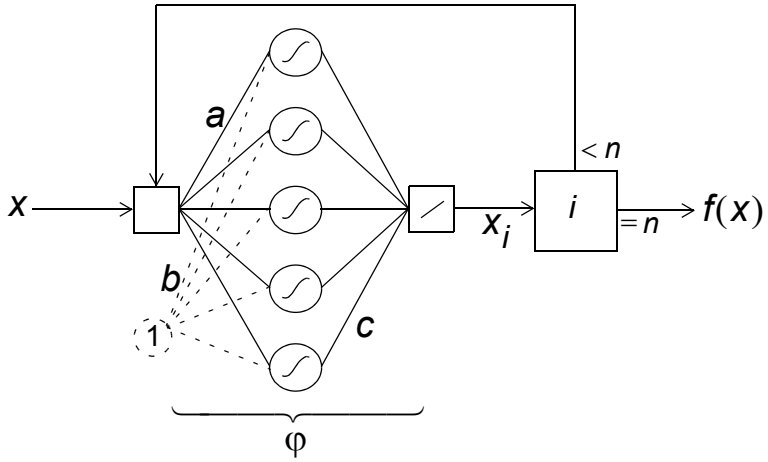

Abbildung 3.5 Iteration eines einzelnen Netzes. Es enthält Gewichte (a) und Biase (b) der verdeckten Schicht und Gewichte der Ausgangsschicht (c).

Wenn G der n-ten Wurzel φ von f entsprechen soll, ist der Netzfehler für einen Eingangswert x gerade $x_n - f(x)$. Als zu minimierende Fehlerfunktion bei einem Trainingsdatensatz aus K Beispielen $({}_k x, {}_k y)$ mit ${}_k y = f({}_k x)$ ergibt sich damit als Summe der quadratischen Fehler

$$E = \frac{1}{2} \sum_{k=1}^{K} ({}_k x_n - {}_k y)^2 ,$$

wobei ${}_k x_n$ den Netzoutput und ${}_k y$ das wahre Ergebnis für das k-te Beispiel im Datensatz bezeichnen.

Zur Minimierung von E mit einem Gradientenverfahren werden die partiellen Ableitungen von E bezüglich der Gewichte a_h, b_h und c_h benötigt:

$$\frac{\partial E}{\partial a_h} = \sum_{k=1}^{K} ({}_k x_n - {}_k y) \frac{\delta_k x_n}{\delta a_h} \tag{3.10}$$

und entsprechend $\frac{\partial E}{\partial b_h}$ und $\frac{\partial E}{\partial c_h}$.
Da die gleichen Gewichte in jedem Iterationsschritt auftauchen, kann der gewöhnliche Backpropagation-Algorithmus nicht angewendet werden, um diese Ableitungen zu berechnen. Stattdessen ist aber ein modifiziertes Verfahren möglich.

Da der Input $x_0 = x$ von den Gewichten unabhängig ist, verschwinden sämtliche partiellen Ableitungen

$$\frac{\partial x_0}{\partial a_h} = 0, \frac{\partial x_0}{\partial b_h} = 0 \text{ und } \frac{\partial x_0}{\partial c_h} = 0$$

und es ist möglich, Rekursionsformeln anzugeben (δ_{jh} bezeichnet das Kronecker-Symbol):

$$\frac{\partial x_{i+1}}{\partial a_h} = \sum_{j=1}^{H} c_j \phi'(a_j x_i + b_j) \left(\delta_{jh} x_i + a_j \frac{\partial x_i}{\partial a_h} \right)$$

$$\frac{\partial x_{i+1}}{\partial b_h} = \sum_{j=1}^{H} c_j \phi'(a_j x_i + b_j) \left(\delta_{jh} + a_j \frac{\partial x_i}{\partial b_h} \right)$$

$$\frac{\partial x_{i+1}}{\partial c_h} = \sum_{j=1}^{H} \left(\delta_{jh} \phi(a_j x_i + b_j) + c_j \phi'(a_j x_i + b_j) \left(a_j \frac{\partial x_i}{\partial c_h} \right) \right)$$

Die wiederholte Anwendung dieser Formeln erlaubt es nun, die für den Minimierungsalgorithmus benötigten Komponenten

$$\frac{\partial x_n}{\partial a_h}, \frac{\partial x_n}{\partial b_h} \text{ und } \frac{\partial x_n}{\partial c_h}$$

und damit entsprechend Gleichung (3.10) die Ableitungen $\frac{\partial E}{\partial a_h}$ usw. zu berechnen. Im normalen Gradientenabstieg nimmt man dann pro Lernschritt als Gewichtsänderung

$$\Delta a_h = \eta \frac{\partial E}{\partial a_h} \text{ usw.} \tag{3.11}$$

Im mehrdimensionalen Fall verwendet man die gleiche Strategie. Allerdings lassen sich die verwendeten Ableitungen nicht mehr übersichtlich darstellen, da eine Ableitung $\frac{\partial X_{i+1,t}}{\partial a_h}$ nicht nur von $\frac{\partial X_{i,t}}{\partial a_h}$, sondern auch von allen $\frac{\partial X_{i,s}}{\partial a_h}$ mit $s \neq t$ abhängt.

Mit den gegebenen partiellen Ableitungen lassen sich auch Gradientenverfahren höherer Ordnung realisieren, die gegenüber Gleichung (3.11) ein erheblich besseres Lernverhalten aufweisen, z. B. Pseudo-Newton Algorithmen wie die Broyden-Fletcher-Goldfarb-Shanno Methode [Bishop 1995].

3.6 Backpropagation through Time

Die Verfahren zur Berechnung iterativer Wurzeln sind verwandt mit Backpropagation through Time, abgekürzt BTT [Waibel et al. 1989, Werbos 1990]. BTT ist eine Methode, rekurrente Netze mit Backpropagation zu trainieren.

Gegeben ist eine Zeitreihe $x_1 ... x_n$. Gesucht ist eine Vorhersage für x_{n+1}. Dabei hängt x_{n+1} unter Umständen nicht nur von x_n, sondern auch von x_{n-1} und noch weiter in der Vergangenheit liegenden Zuständen ab. Für die Modellierung eines solchen Systems werden rekurrente Netze ähnlich Abbildung 3.5 eingesetzt [Elman 1990]. Backpropagation through Time entfaltet ein solches Netz in die Hintereinanderschaltung von reinen feed-forward Netzen ähnlich Abbildung 3.4 mit jeweils gleichen Gewichten und nutzt normales Backpropagation, um die Ableitungen der Gewichte in den einzelnen Teilnetzen zu bestimmen. Es wird also ebenfalls ein Modell für einen einzelnen Zeitschritt bestimmt, das x_i in x_{i+1} transformiert. Der Unterschied zur den hier verwendeten Methoden bei der Berechnung von iterativen Wurzeln ist allerdings, dass BTT die Kenntnis aller x_i vorraussetzt.

Eine Kombination von BTT und iterativen Wurzeln könnte dazu verwendet werden, Zeitreihen mit höherer Auflösung zu modellieren, als durch die Daten vorgegeben.

3.7 Vorprägung des iterierten Netzes

Funktionswurzeln sind oft nicht eindeutig bestimmt. Die iterierten Netze finden irgendeine der möglichen Lösungen. Allerdings sind im Anwendungsfall meist nur „reguläre" Lösung gesucht. Es besteht die Möglichkeit, mit verschiedenen Initialisierungen das Training zu wiederholen und darauf zu warten, dass sich eine Lösung der gewünschten Art einstellt. Allerdings besteht bei einigen Beispielfunktionen die Tendenz, fast nur auf nicht-reguläre Lösungen zu konvergieren.

So gibt es als einfachstes Beispiel nur eine einzige lineare monoton steigende Wurzel von $f(x) = x$, nämlich x und dagegen unendlich viele monoton fallende: $a - x$ mit beliebigem a. Ein Verfahren, das von beliebigen Startwerten ausgeht, wird daher deutlich öfter die fallende Lösung finden. Entsprechendes gilt für alle monoton steigenden Funktionen.

Daher ist eine Steuerung des Lernprozesses in Richtung auf gewünschte Ergebnisse von Vorteil. Geht es z.B. um eine monoton steigende Funktion, so ist in aller Regel auch nur die monoton steigende Lösung interessant. Hier könnte man einen weiteren Fehlerterm einführen, der unerwünschte Verhaltensweisen „bestraft“, d.h. in diesem Falle bei negativen Steigungen groß wird:

$$\sum_i \left(\left| \frac{\varphi(x_i) - \varphi(x_{i+1})}{x_i - x_{i+1}} \right| - \frac{\varphi(x_i) - \varphi(x_{i+1})}{x_i - x_{i+1}} \right)^2,$$

summiert über alle sortierten Eingangswertpaare (x_i, x_{i+1}) wäre in diesem Fall eine solche monotone, stetige und differenzierbare Fehlerfunktion.

Solche Regularisierungen sind jedoch problematisch, da es i.a. schwierig ist, einen stetigen Ausdruck zu finden, der bei Erfüllung der Bedingung auf Null gehen kann, ohne die anderen Zielvorgaben - genaue Anpassung der Funktion und Identität der Teilnetze - zu degradieren. Es ist entscheidend, ein Fehlermaß zu verwenden, das theoretisch in allen Termen Null erreichen kann. Andernfalls tritt eine Konkurrenzsituation auf, die zu nicht optimaler Funktionsanpassung bzw. ungleichen Teilnetzen führen kann.

Eine andere Möglichkeit, den Gradientenabstieg auf die gewünschte Lösung hinzuführen, besteht darin, bereits in der Nähe dieser Lösung zu starten. Dazu muss man eine der gesuchten Lösung möglichst ähnliche Funktion „erraten“ und jedes Teilnetz darauf trainieren. So erhält man ein Netz, das grob die Aufgabe erfüllt. Startet man jetzt eine gradientenbasierte Optimierung, ist die Wahrscheinlichkeit hoch, eine Lösung, die der Vorgabe möglichst ähnlich ist, zu finden.

Allerdings ist es schwierig, im allgemeinen Fall eine der regulären Funktionswurzel ähnliche Funktion zu „erraten“. Bei monoton steigenden Funktionen gilt, dass auch alle Wurzeln monoton steigend sind und der Graph zwischen $f(x)$ und der Winkelhalbierenden $y = x$ liegt. Daher sind die folgenden Start-Funktionen geeignete Initialisierungen für die Teilnetze.

3.7.1 Identitätsfunktion

Für $n \to \infty$ konvergiert die Funktionswurzel $f^{1/n}$ von monoton steigenden Funktionen f gegen die Identitätsfunktion $f^0 = id$. Somit ist besonders bei höheren Wurzeln monoton steigender Funktionen die Identität ein guter Startpunkt.

3.7.2 Zielfunktion f(x)

In vielen Fällen haben iterative Wurzeln eine ähnliche Form wie die Funktion selbst. Dies ist bei monoton steigenden Funktionen der Fall, aber oft auch bei symmetrischen Funktionen. In diesen Fällen ist die Funktion selbst eine sinnvolle Initialisierung.

3.7.3 Graph zwischen f(x) und y=x

Bei monoton steigenden Funktionen $f(x)$ liegt die Wurzel stets zwischen dem Graphen von f und dem von $y = x$. Daher ist in diesen Fällen die Funktion $\frac{f(x)-x}{2}$, deren Graph genau in der Mitte verläuft, eine gute Anfangsnäherung. Bei n-ten Wurzeln liegt der Graph noch weiter an der Winkelhalbierenden, dort ist

$$g(x) = \frac{1}{n}(f(x) + (n-1)x) \tag{3.12}$$

eine geeignete Startfunktion. Abbildung A.4 auf Seite 98 demonstriert dies am Beispiel e^x.

3.8 Praktisches Vorgehen

Um abschätzen zu können, inwieweit das Problem überhaupt mit einem neuronalen Netz gelöst werden kann, empfiehlt es sich, zunächst nur die Funktion $f(x)$ selbst zu modellieren. Dabei verwende man prinzipiell die gleichen Methoden, die man später auch für die iterative Modellierung zur Verfügung hat.

Sind die Trainingsdaten nicht „perfekt“, z.B. Rechenwerte aus einer genau berechneten Tabelle der zu betrachtenden Funktion, sondern rauschbehaftete Messwerte, dann empfiehlt es sich, das für Funktionsapproximationen auch sonst übliche Repertoire, wie z.B. Crossvalidation, aufzubieten, um Overfitting zu verhindern.

Zu beachten ist dabei unbedingt, dass keine unabhängige Skalierung der Ein- und Ausgänge vorgenommen werden darf. Viele Simulatoren nehmen solche Skalierungen automatisch vor, weil das eine der wichtigsten Methoden der Datenvorverarbeitung ist, um gute Ergebnisse mit neuronalen Netzen zu erzielen [LeCun et al. 1998]. Dies muss jedoch vermieden werden. Da die verallgemeinerte Iteration nur invariant gegenüber Transformationen *aller* Achsen gleichzeitig ist (siehe Abschnitt 2.6.2), dürfen die Ein- und Ausgänge des Netzes nur gemeinsam skaliert und verschoben werden.

Ferner empfiehlt es sich, in der Ausgabeschicht und allen „Datenschichten“ lineare Neuronen zu verwenden, um Beschränkungen des Wertebereiches durch Sättigung zu vermeiden. Auf jeden Fall müssen die Datenschichten den gleichen Neuronentyp verwenden wie die Ausgabeschicht.

Die Eingabeneuronen dürfen keinen Bias haben. Bei den Zwischenschichten und der Ausgabeschicht muss gelten, dass entweder alle oder gar keine Schicht einen Biaseingang haben.

Wichtig ist, eine minimale Netzgröße zu finden, mit der die Funktion *f* zufriedenstellend angenähert werden kann. Dies reduziert die Wahrscheinlichkeit, später nicht reguläre Wurzeln zu finden, z.B. oszillierende Lösungen.

Es gibt keine allgemeingültigen Gesetzmäßigkeiten über die Komplexität von iterativen Wurzeln im Vergleich zur Originalfunktion: Wurzeln von Iterierten der logistischen Abbildung haben bis zu dreimal weniger lokale Extrema als die Originalfunktion, sind damit als einfacher anzusehen. Die iterative Wurzel des Bernoulli-Shiftes $\left(\frac{1}{2}, \frac{1}{2}\right)$ [Krengel & Michel 1967, Ornstein 1974], der selbst überall bis auf eine Nullmenge stetig ist, ist dagegen *überall unstetig*, also sicher als komplexer anzusehen. Jedoch kann man In der Regel bei Anwendungsproblemen von einer ähnlichen Komplexität wie bei der Originalfunktion ausgehen. Daher ist ein Netz, das die Funktion *f* gut annähern kann auch ein guter Kandidat für ein Teilnetz, das deren iterative Wurzel modellieren soll.

Modellierungen von Systemen mit mehreren Ein- und Ausgängen sind als sehr experimentell anzusehen, es gibt kaum konkrete mathematische Aussagen zu iterativen Wurzeln von höherdimensionalen Abbildungen. Die mathematische Literatur konzentriert sich einerseits auf iterative Wurzeln reell- und komplexwertiger Funktionen; einige der oben angegebenen Instruktionen basieren auf deren bekannten Eigenschaften. Außerdem gibt es Untersuchungen zur fraktionalen Iteration von völlig abstrakten Abbildungen, aus denen jedoch kaum konkrete Anweisungen zu deren Lösung abgeleitet werden können [Targonski 1981].

3.9 Beispiele

3.9.1 f(x)=x^2

Hier werden die 4-te iterative Wurzel der Funktion $f(x) = x^2$ und die fraktionalen Iterationen $f^{m/4}$ bestimmt. Abbildung 3.6 B zeigt eine Lösung, die das Netz gefunden hat und sehr gut mit der in diesem Fall analytisch bestimmbaren Lösung $f^{1/2}(x) = |x|^{\sqrt{2}}$ übereinstimmt. In A sind alle fraktionalen Iterationen f^{-1} bis f^2 in Schritten von $1/4$ dargestellt. Die Inversen wurden mit einem zweiten, identischen Netz berechnet.

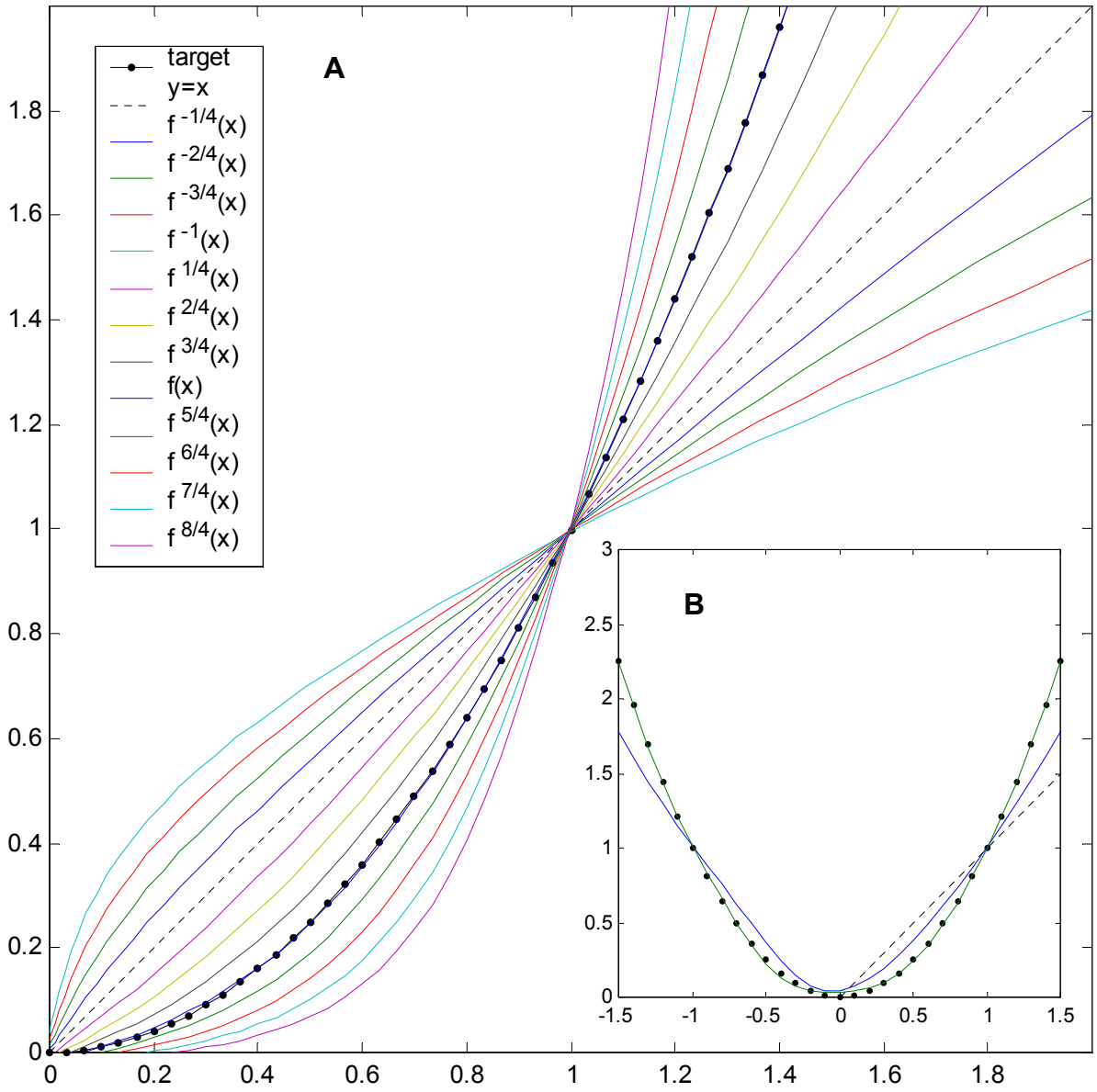

Abbildung 3.6 Fraktionale Iterationen von $f(x)=x^2$. Die Wurzeln und Inversen wurden mit einer $(1-5-1)^4$ Architektur berechnet. Kleiner Graph B: Die symmetrische Lösung auch für negative x.

Abbildung 3.7 zeigt weitere Lösungen, die vom gleichen Netz bei anderen Initialisierungen gefunden werden. A entspricht einer weiteren formal möglichen iterativen Wurzel von *f*. Lediglich im Zentrum hat die wirkliche Wurzel kein Maximum, sondern einen Pol. Aber innerhalb der begrenzten Trainingsdaten aus dem Intervall (-2,2) ist die gefundene Lösung konsistent. Dies trifft auch auf B zu, allerdings gibt es hier keine korrespondierende Lösung auf ganz $\mathbb{R}$. Die gefundene Lösung lässt sich nicht auf einen größeren Definitionsbereich erweitern. Diese „Lösung“ ist völlig beliebig: Ein Intervall wird auf ein zweites, nicht überlappendes Intervall abgebildet.

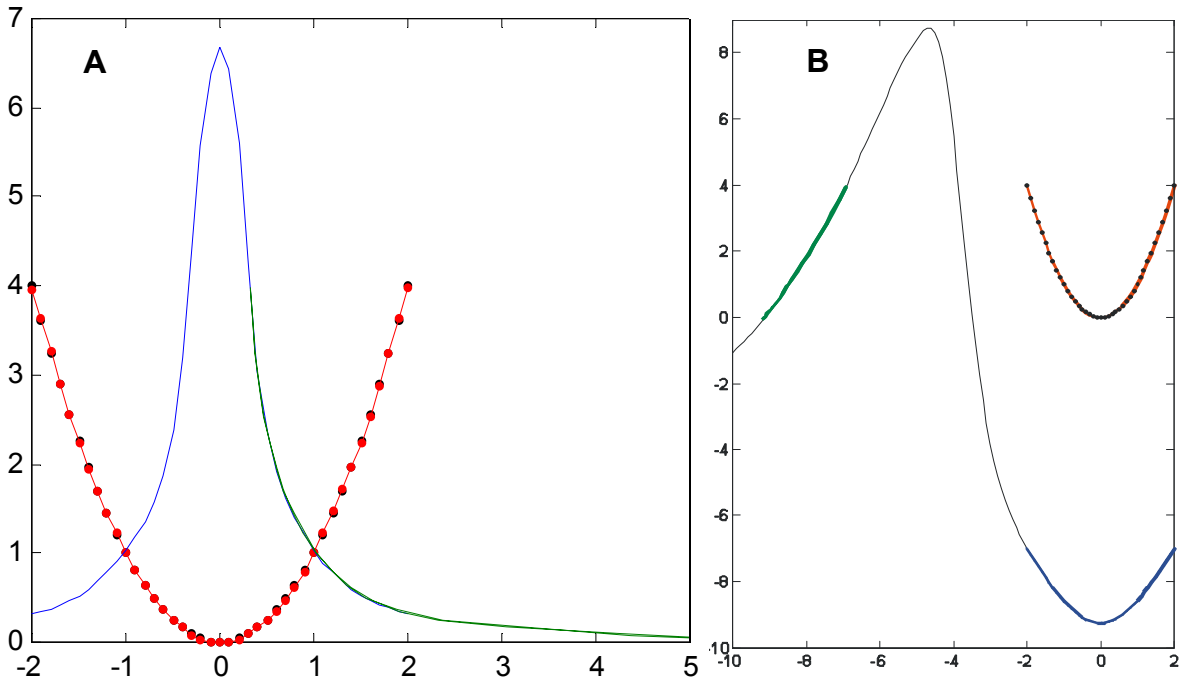

Abbildung 3.7 Andere Lösungen für $f(x)=x^2$. A: Eine fast „echte“ iterative Wurzel B: Eine falsche Lösung, die aus nicht überlappenden Intervallen resultiert. Das erste Teilnetz wird nur im unteren durchgezogenen Bereich aktiviert, das zweite entsprechend oben links.

3.9.2 Die Wurzeln der Rotation

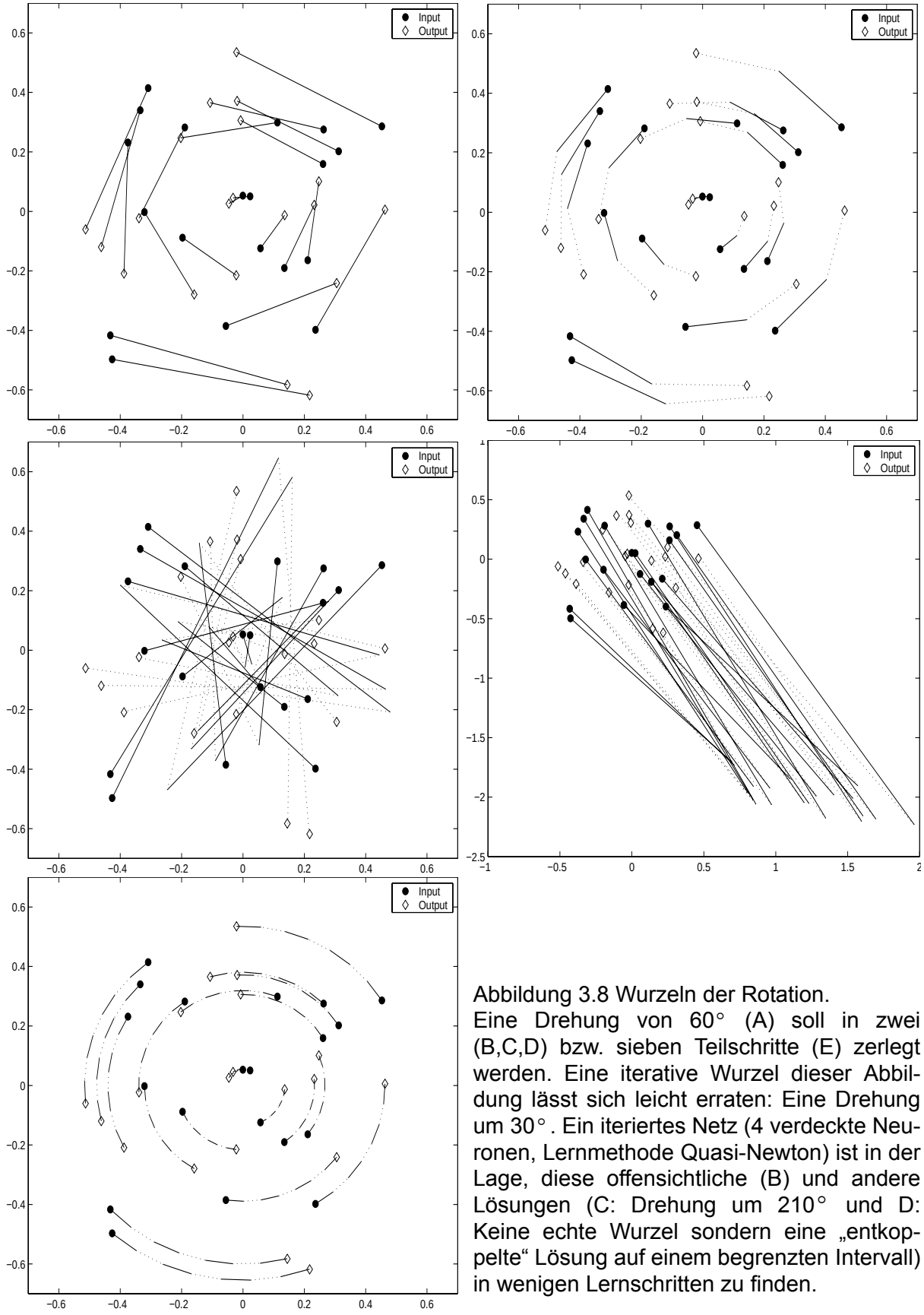

Abbildung 3.8 Wurzeln der Rotation.
Eine Drehung von 60° (A) soll in zwei (B,C,D) bzw. sieben Teilschritte (E) zerlegt werden. Eine iterative Wurzel dieser Abbildung lässt sich leicht erraten: Eine Drehung um 30°. Ein iteriertes Netz (4 verdeckte Neuronen, Lernmethode Quasi-Newton) ist in der Lage, diese offensichtliche (B) und andere Lösungen (C: Drehung um 210° und D: Keine echte Wurzel sondern eine „entkoppelte“ Lösung auf einem begrenzten Intervall) in wenigen Lernschritten zu finden.

KAPITEL 4

Zwischenprofilmodellierung beim Stahlwalzen

Das im Folgenden dargestellte Problem gab den Anstoß zur Entwicklung der neuronalen Methoden zur Berechnung von fraktionalen Iterationen. Ein Industrieunternehmen, das Steuerungen für Anlagen zur Stahlerzeugung herstellt, setzt seit geraumer Zeit erfolgreich neuronale Netze zur Modellierung von Produktionsprozessen ein. Sie werden vor allem in der Level II Automatisierung genutzt, also nicht in der direkten Echtzeitregelung des Prozesses, sondern in Produktionsplanung und Optimierung. Auch für die Modellierung bestimmter Eigenschaften des Breitbandwarmwalzprozesses zeigten sich neuronale Netze, was die Genauigkeit betrifft, den bisher verwendeten physikalisch mathematischen Modellierungen überlegen. Daher bestand der Wunsch, auch die Modellierung des Bandprofils mit neuronalen Methoden durchzuführen.

Doch zusätzlich zu üblichen Anforderungen zur Modellbildung trat hier eine Fragestellung auf, die über den Stand der Technik hinausging, sowohl in der theoretischen Beschreibung als auch hinsichtlich konkret anwendbarer Verfahren. Es wird sich zeigen, dass fraktionale Iteration die Grundlage der datenbasierten Modellbildung dieses Prozesses darstellt.

Im Betrieb einer Anlage fallen laufend Daten an, die den Zusammenhang zwischen Eingangsgrößen und Ergebnis dokumentieren. Eingangsgrößen sind zum einen gegebene Materialeigenschaften und Prozessparameter, zum anderen gezielt veränderbare Stellgrößen. Allerdings ist dieser Zusammenhang nicht als absolut fest anzusehen: Nicht alle auf das Endergebnis wirkenden Einflussgrößen sind bekannt oder können erfasst werden. Materialabnutzung, Toleranzen in Verbrauchsteilen, sogar Einflüsse des Wetters etc. können auch bei einer identischen Einstellung aller Maschinenparameter zu einem späteren Zeitpunkt zu abweichenden Resultaten führen. Betreiber der Fabriken sprechen gerne von der „Tagesform“ ihrer Anlage.

Daher sind adaptive Modelle nötig, die in der Lage sind, diesem driftenden Prozess zu folgen. Eine genaue physikalische Beschreibung dieser Einflüsse ist in der Regel nur

begrenzt möglich, so besteht das klassische Verfahren darin, zu den physikalisch fundierten Modellgleichungen noch adaptive Terme hinzuzufügen, die dann durch Optimierungsverfahren kontinuierlich oder in regelmäßigen Abständen angepasst werden.

Neuronale Netze bieten sich hier aufgrund ihrer Lernfähigkeit an, allerdings ist die Problematik des „kontinuierlichen Mitlernens“ noch nicht abschließend gelöst, sondern Gegenstand aktueller Forschung [Protzel 1998]. Trotzdem sind inzwischen adaptive Netze in einer Reihe von modernen Walzstraßen im Einsatz [Neumerkel 1996, Martinez 1995, Ordieres 1996]. Allen diesen Anwendungen ist jedoch gemein, dass der tatsächliche Wert der zu modellierenden Größe irgendwann zur Verfügung steht. Damit sind diese Anwendungen Beispiele für klassisches überwachtes Lernen.

Bei dem Problem der Zwischenprofilmodellierung besteht das zu prognostizierende Resultat jedoch nicht nur aus dem nachmessbaren Endwert, sondern für die Prozessführung ist auch der prozessinterne Verlauf dieser Größe wichtig. Tatsächliche Messwerte dieser „Zwischenprofile“ stehen aber aus technischen Gründen nicht zur Verfügung. Eine reine „Black-box“, die nur das Endprofil perfekt vorhersagt, reicht nicht aus, man braucht ein Modell, das auch Fragen anderer Art beantworten kann, als sie durch die Trainingsdaten vorgegeben sind.

4.1 Walzen von Stahl

Eine moderne Walzstraße gehört zu den größten und aufwendigsten Anlagen in der Stahlindustrie. Kosten von bis zu 750.000.000 € für eine Installation sind keine Seltenheit. Stahlbrammen von einigen Zentimetern Dicke werden zu millimeterdünnem Blech ausgewalzt. Pro Minute können bis zu 10 Tonnen verarbeitet werden. Dabei gelten rigorose Güteanforderungen an das Produkt. Toleranzen von wenigen μ sind von den Kunden, z.B. Automobilherstellern, vorgeschrieben. Der Walzvorgang für eine Bramme dauert nur einige Sekunden. Ist die Anlage nicht optimal eingestellt, kann das gesamte Ergebnis zum Alteisen gegeben werden und muss erneut eingeschmolzen werden. Müssen Einstellungen im laufenden Betrieb vorgenommen oder verändert werden, ist dies mit hohen Kosten verbunden.

Daher ist es wichtig, Regel- und Stellgrößen schon bevor ein Band in die Anlage läuft, für jedes Werkstück optimal voreinzustellen. Zu diesem Zweck verwenden die Betreiber der Anlage ein Prozessmodell, das es erlaubt, verschiedene Walzstrategien offline durchzuspielen und geeignete Einstellungen für jedes Werkstück zu finden [Lindhoff 1994].

Die Eigenschaften eines Stahlbandes sind nicht nur durch Breite und Dicke bestimmt. Profil und Planheit des Bandes werden genauso vom Walzprozess beeinflusst, wie Materialeigenschaften des Stahls. Walzen ist praktisch kontinuierliches Schmieden. Ein einmaliges Walzen des Bleches reicht nicht aus, um alle gewünschten Eigenschaften zu formen. Früher und in manchen „Billiganlagen“ noch heute wurde das

Blech mehrfach hin und her durch dasselbe Walzgerüst geführt. Moderne Anlagen bestehen aus bis zu sieben identischen Gerüsten direkt hintereinander. Der Walzvorgang kann also als iterierter Prozess verstanden werden, wobei allerdings die einzelnen Gerüste unterschiedliche Einstellungen haben können. Die Verteilung der Walzkräfte und anderer Stellgrößen auf die einzelnen Gerüste bezeichnet man als Stichplan.

Der Erstellung dieses Stichplans dient die Vorausberechnung der zu erwartenden Endergebnisse wie Dicke, Planheit, Profil, Temperatur.

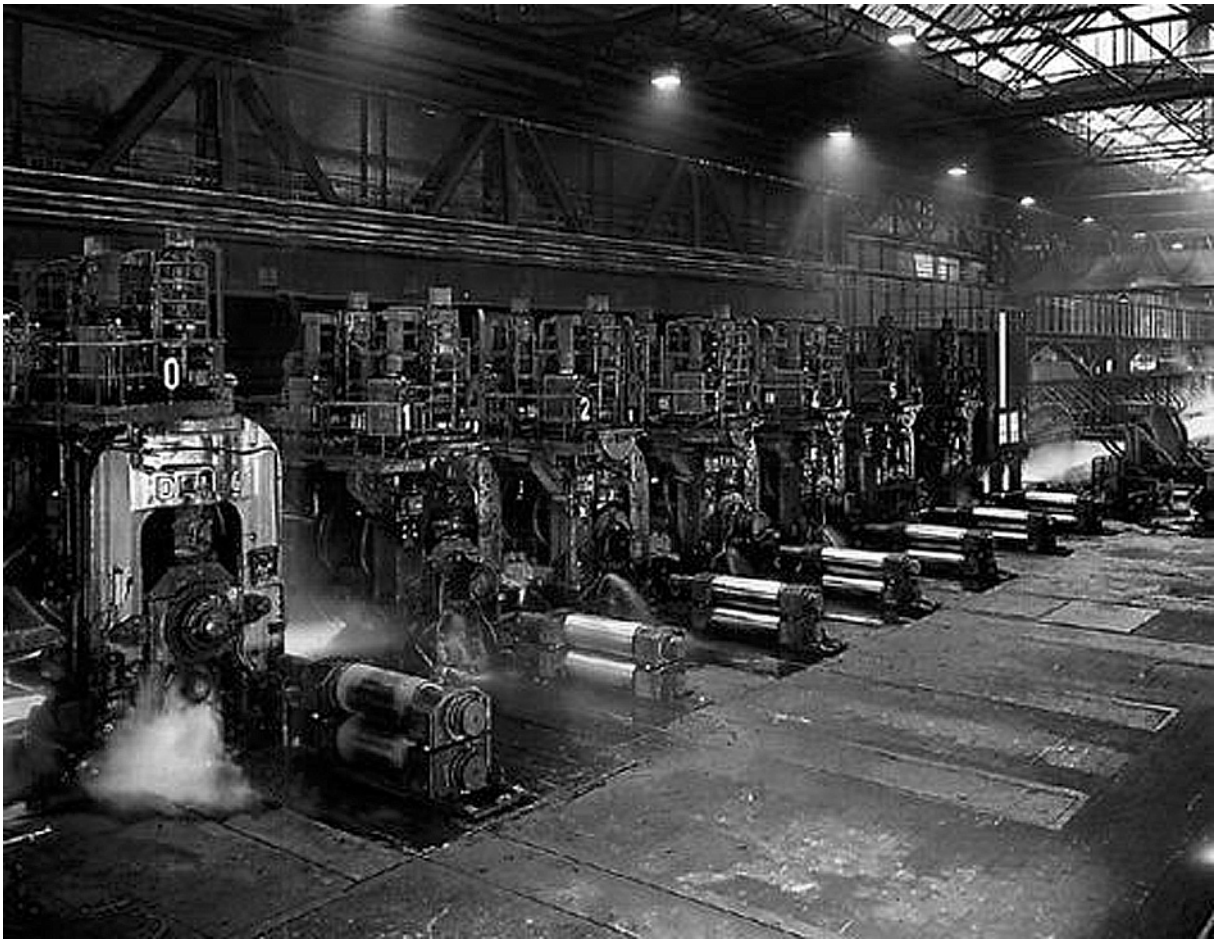

Abbildung 4.1 7-gerüstige Breitbandwarmwalzstraße. Foto mit freundlicher Genehmigung der SIEMENS AG.

4.2 Zwischenprofile

Neben der Breite und der Dicke ist das Profil ein wichtiges Maß für die Geometrie des Walzgutes nach Verlassen der Fertigstraße. Zur numerischen Behandlung ist das Profil vereinfacht definiert als Dickenunterschied zwischen der Mitte und den Rändern eines Bandes, wie in Abbildung 4.2 gezeigt:

$$p = h - \frac{h_1 + h_2}{2}$$

Bei konvexen Oberflächen wird das Profil positiv, bei konkaven negativ. Es wird Null, wenn die drei Werte auf einer Geraden liegen, unabhängig davon, welchen Anstieg diese Gerade besitzt. Somit werden stark unterschiedliche Formen nicht unterschieden: Ein Profilwert von Null kann einen rechteckigen, dreieckigen oder trapezförmigen Bandquerschnitt beschreiben. In der Praxis werden allerdings symmetrische Profile angenommen [Martinez 1995].

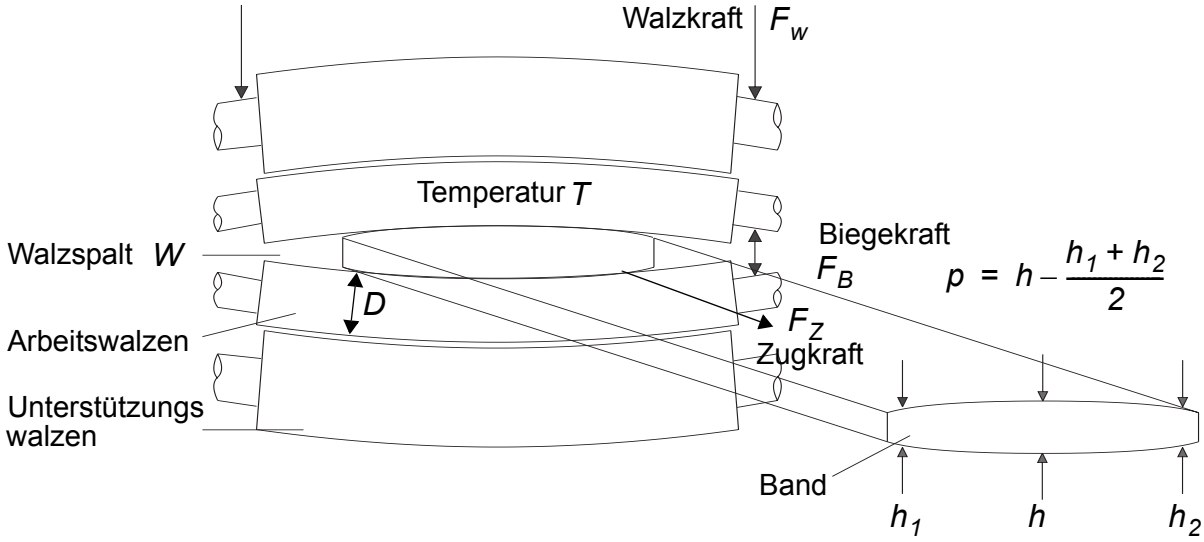

Abbildung 4.2 Definition des Walzbandprofils p.

Das Ziel ist, das Profil während des Warmwalzens so klein wie möglich zu halten, da es in späteren Kaltwalzprozessen nicht mehr beeinflusst werden kann. Das Profil des Walzgutes nach dem Durchlaufen eines Gerüstes ergibt sich als Funktion der Eigenschaften des Bandes beim Eintritt in das Gerüst und dem aktuellen Zustand und den Einstellungen des Gerüstes. Abbildung 4.3 zeigt den schematischen Aufbau der Fertigstraße aus Abbildung 4.1 mit einigen Einflussgrößen auf das Profil.

Das dem Stahl aufgeprägte Walzspaltprofil resultiert aus einer mechanischen und einer thermischen Walzenverformung und spiegelt die Prozessvergangenheit des Walzvorgangs wider. Die thermische Walzenverformung (thermischer Crown) hängt z.B. von den Temperaturen, Breiten und Pausenzeiten einer ganzen Reihe aufeinanderfolgender Bänder ab.

Die existierenden mathematischen Modelle, beispielsweise zur Berechnung des Walzspaltprofils, beruhen zwar auf der Modellierung der physikalischen Zusammenhänge, jedoch sind in der Praxis viele idealisierte Voraussetzungen für die Gültigkeit der Modelle nicht erfüllt. Dies führt zu der Verwendung von zusätzlichen, heuristischen Modellen mit einer Vielzahl unbekannter Parameter, deren optimale Einstellung gefunden werden muss. Ein besonderes Problem liegt dabei in der Struktur der Profil-

modelle. Während des gesamten Walzprozesses ändert sich das Profil des einlaufenden Stahlbandes von Gerüst zu Gerüst. Erst am Ende der Walzstraße kann es gemessen werden und mit den Vorhersagen des Modells verglichen werden. Das Profil des Bandes zwischen den Gerüsten ist aus technischen Gründen nicht direkt messbar, jedoch benötigt man für die Stichplanberechnung eine Prognose für den Verlauf des Bandprofils über der gesamten Walzstraße.

Dieser Profilverlauf wird bisher durch das n-fache Hintereinanderschalten eines mathematischen Gerüstmodells erzeugt. Es hat sich jedoch gezeigt, dass kontinuierlich lernende Neuronale Netze in der Lage sind, das Endprofil insbesondere bei Produktumstellungen um bis zu 20% genauer vorherzusagen [Protzel 2000]. Da das neuronale Netz wiederum keine Zwischenprofile erzeugt, ist es wünschenswert, die Eigenschaften von mathematischem Modell und neuronalem Netz in einem Hybridsystem zu verbinden.

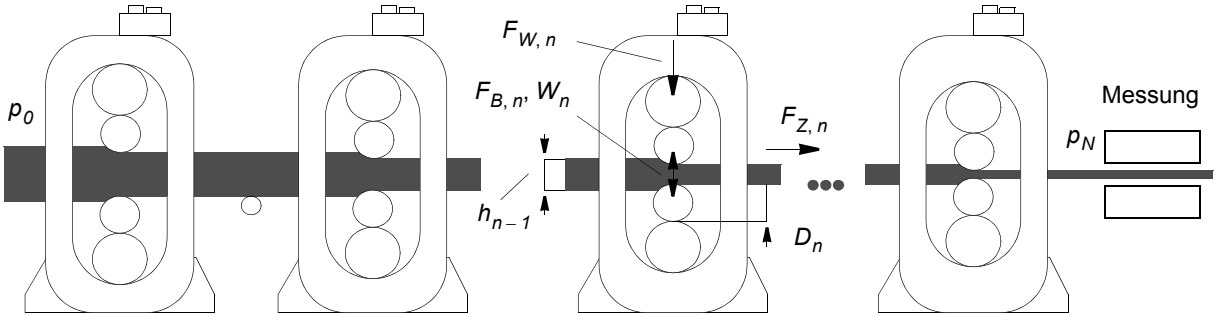

Abbildung 4.3 Schema des Profilverlaufes in einer N-gerüstigen Walzstraße. Von links läuft eine Stahlbramme der Dicke h_0 ein und wird sukzessive in mehreren Schritten von N technisch identischen Walzgerüsten auf ein Blech der Enddicke h_N gewalzt. Neben der Dicke werden beim Walzen auch andere geometrische Eigenschaften und Materialeigenschaften des Bleches beeinflusst, darunter das hier zu modellierende Profil.

Gegeben: Profil des Bandes am Eingang (p_0) und am Ausgang (p_N) der Walzstraße

Parameter der Einzelgerüste: $n = 1, \ldots, N$:

h_{n-1}, h_n	Banddicke am Ein/Ausgang des Gerüstes n
D_n	Durchmesser der Arbeitswalze
B_n	Bandbreite (wird als konstant angesetzt: $B_n = B \forall n$)
W_n	Walzspaltprofil
$F_{W,n}, F_{B,n}, F_{z,n}$	Walzkraft, Biegekraft, Zugkraft
T_n	Walzentemperatur
S_n	Walzenschiebeposition
L	Materialkenngröße (Legierungskennung)

Gesucht: Modell des Einzelgerüstes; Zwischenprofile $\hat{p}_1, \ldots, \hat{p}_{N-1}$ und Endprofil $\hat{p}_N$

4.3 Der Stand der Technik - Profilmodell

Die Aufgabenstellung bei der Vorhersage der Zwischenprofile ist in Abbildung 4.3 zusammengestellt. Die verschiedenen an einem Gerüst auftretenden Teilprozesse sind stark ineinander verzahnt. Mit Ausnahme der am Ein- und Ausgang der Walzstraße messbaren Größen, der einzustellenden Kräfte und der Durchmesser der Arbeitswalzen werden alle für die Zwischenprofilvorhersage relevanten Größen aus einer Voraus– bzw. einer Nachberechnung, d.h. über weitere Modelle, erhalten. Die Vorausberechnung prognostiziert die jeweilige Prozessgröße anhand gegebener Einstellungen der sie beeinflussenden Größen. Die Nachberechnung korrigiert diese Werte entsprechend einer Differenz in der von ihnen beeinflußten Größen zu deren Istwert(en).

4.3.1 Modellgleichungen und Parameter

Profilbestimmung und -steuerung gehen davon aus, dass beim Durchgang eines Bandes durch ein Gerüst das Walzspaltprofil (die Kontur der Arbeitswalzen im Walzspalt) komplett auf das Band übertragen wird; demzufolge ist das Austrittsprofil des Bandes identisch dem Walzspaltprofil. Die korrekte Bestimmung des Walzspaltprofils setzt die Kenntnis der Walzkraftverteilung über die Ballenlänge voraus. In Modellen wird sie für ein Vier–Walzen–Gerüst mit je zwei Arbeits- und Stützwalzen über einen Iterationsalgorithmus bestimmt:

Ausgegangen wird von einer konstanten Walzkraftverteilung, die dann bis zur Konvergenz durch eine geschlossene Kette von Modellen modifiziert wird [Riekmann 1989]:

Walzkraftverteilung $\Rightarrow$ Elastische Verformung des Walzensatzes $\Rightarrow$ Kontur/Profil des auslaufenden Bandes $\Rightarrow$ Ein- und auslaufende Banddickenverteilung $\Rightarrow$ Zugspannungsverteilung, Fließkurve, Reibungszahl $\Rightarrow$ Walzkraftverteilung.

Das Walzspaltprofil selbst wird aus einer weiteren Iterationsroutine ermittelt, die für die berechnete Walzkraftverteilung zunächst Biegung und Abplattung der Arbeitswalzen bestimmt. Für zwei benachbarte Walzen gilt, dass sie sich in der gemeinsamen Kontaktzone so verformen, dass die entstehenden Kraftverteilungen zu äußeren Kräften und Momenten im Gleichgewicht stehen. Die Verformungen müssen geometrisch kompatibel sein – Walzen können sich nicht durchdringen. Ist diese Bedingung nicht eingehalten, wird die Abplattung korrigiert und eine neue Verteilung der Zwischenkraft zwischen Arbeits- und Stützwalze bestimmt.

Das Walzspaltprofil ergibt sich aus der Superposition von Biegelinie, Abplattung und Schliff der Arbeitswalze. Beim Warmwalzen wird zusätzlich ein Temperaturmodell nötig, das den „thermischen Crown“ berücksichtigt. Dies ist ein Dickenunterschied zwischen Walzenmitte und -rand, der aus der stärkeren Erwärmung der Walzenmitte durch die glühenden Stahlbänder, die schmaler sind als die Walzenbreite, resultiert.

Ist auch nur eine der Komponenten dieses Modells unzureichend bekannt, kann sich eine starke Abweichung zum real vorhandenen Walzspalt ergeben – Austrittsprofil des Bandes und berechnetes Walzspaltprofil stimmen nicht überein.

Daher wird zusätzlich eine empirische Übertragungsfunktion verwendet, die diese Abweichungen durch Einbeziehung des einlaufenden Bandprofiles zu kompensieren versucht. Die Profilvorausberechnung für ein Profil p_n am Ausgang des Gerüstes n erfolgt anhand des Gleichungssystems

$$
\begin{aligned}
p_n &= k_n \cdot \frac{h_n}{h_{n-1}} \cdot p_{n-1} + (1 - k_n) \cdot W_n + c_{n3} \cdot a \cdot h_n \\
k_n &= \frac{1}{\pi} atan(g_n) + \frac{1}{2} \\
g_n &= \frac{c_{n1} - \ln x_n}{c_{n2}} \\
x_n &= \frac{\sqrt{D_n} \cdot h_n^{1,5}}{B^2}
\end{aligned}
\tag{4.1}
$$

p_n setzt sich aus einer Linearkombination von Walzspaltprofil W_n und Banddickenprofil p_{n-1} sowie einem adaptiven Term zusammen. Den Grad des Einflusses beider Profile in der Linearkombination bestimmt der Faktor k_n. Er ist eine nichtlineare Funktion, die neben den Prozessvariablen bzw. Parametern der Gerüste (Arbeitswalzenradius D, Banddicke h und Bandbreite B) zwei Koeffizienten c_{n1} und c_{n2} enthält, die zusätzliche Variabilität in die Übertragungsfunktion der Gerüste einbringen können.

Diese Koeffizienten c_{ni} werden offline durch periodisch durchgeführte Optimierungsverfahren an Datensätze aus dem Betrieb der Anlage angepasst. Der Adaptionskoeffizient a, aus dem Term $c_{n3} \cdot a \cdot h_n$, wird permanent online aus der Folge vorangegangener Endprofile berechnet und dient zur sofortigen Verminderung des Modellfehlers.

4.4 Modellierung des Endprofils mit neuronalen Netzen

Eine Reihe von Modellierungen im Bereich von Walzwerken werden heute mit adaptiven neuronalen Netzen durchgeführt. Sie haben sich als z. Zt. bestes verfügbares Werkzeug erwiesen, die problematische Kombination aus Nichtlinearität, Instationarität und den Mangel an exakten Modellen zu bewältigen, die einen so komplexen Prozess begleitet.

Anhand von Produktionsdaten konnte gezeigt werden, dass auch das Bandprofil von adaptiven neuronalen Netzen deutlich genauer vorausberechnet werden kann, als

von dem eben beschriebenen Profilmodell, das bisher allgemein Verwendung findet. Besonders bei den sogenannten Umstellungsbändern, das sind die Bleche, die sich stark vom direkt vorher gefahrenen Produkt unterscheiden, werden deutlich geringere Fehler möglich (Tabelle 4.1).

Dies ist ein entscheidender Vorteil, da die Prozessregelung hier noch nicht greift. Die Totzeit, bis das Endprofil erfasst werden kann und entsprechende Stellgrößen adaptiert werden können, entspricht der Durchlaufzeit durch die gesamte Straße. Fehler, die bei der Vorausberechnung und im Stichplan gemacht wurden, wirken sich in dieser Zeit voll aus und können schlimmstenfalls dazu führen, dass mehrere hundert Meter Stahlblech zu Ausschuss werden.

Neuronale Algorithmen sind bereits in den Anlagen integriert, z.B für die Modellierung der Walzkraftverteilung. Daher ist die Implementierung einer Prognose des Endprofils ohne großen Aufwand möglich. Aufgrund der Stabilität werden vor allem lokale lineare Karten (local linear maps) eingesetzt [Martinez et.al. 1995].

4.5 Modellierung der Walzstraße mit iterierten Netzen

In Kapitel 3 wurden Methoden entwickelt, die die Modellierung von parametrisierten Funktionswurzeln entsprechend Abbildung 2.4 auf Seite 44 mit neuronalen Netzen ermöglichen. Das Modell (4.1) stellt solch einen Zusammenhang dar:

$$p_n = f(p_{n-1}, h_n, W_n, D_n)$$

Dies lässt sich in ein neuronales Netz entsprechend Abbildung 4.4 übersetzen.

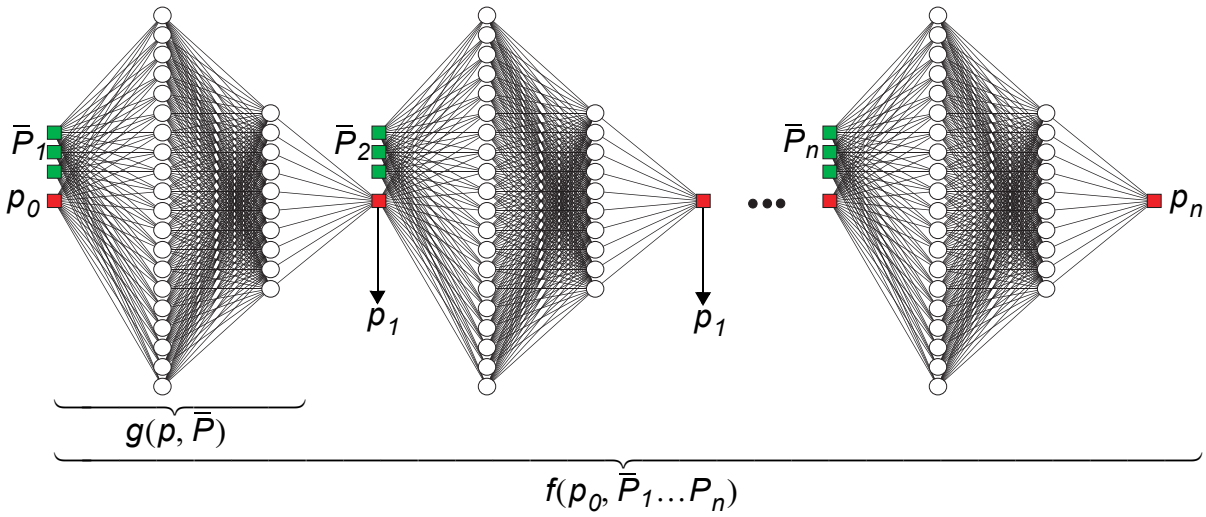

Abbildung 4.4 Iteriertes Netz zur Modellierung von Zwischenprofilen. Jedes Teilnetz entspricht einem Gerüst $p_{out} = g(p_{in}, \bar{P})$, die Gewichte der Teilnetze sind identisch. $\bar{P}$ ist ein Satz von Einstellungen, die für jedes Gerüst unterschiedlich sein können.

Aufgrund der Entstehung der Daten durch einen entsprechenden iterierten Prozess kann man von der Existenz einer Funktion g ausgehen, ob g jedoch eindeutig rekonstruierbar ist, ist ungeklärt. Das Problem ist zu komplex, als dass man einen der Sätze über die Eindeutigkeit von Funktionswurzeln sinnvoll anwenden könnte. Daher wurde experimentell versucht, mit verschiedenen Netzen entsprechend Abbildung 4.4 Zwischenprofilverläufe zu rekonstruieren. Allerdings führte diese Untersuchung auf den vorhandenen Daten zu keinem befriedigenden Ergebnis:
Obwohl das Endprofil durch ein solches Netz gut modelliert werden kann, sind die Lösungen für die Einzelgerüstfunktion g nicht reproduzierbar. Verschiedene Topologien des Teilnetzes und sogar jeweils andere Initialisierungen der Gewichte führen zu sehr unterschiedlichen Profilverläufen (Abbildung 4.5), d.h. unterschiedlichen Gerüstmodellen.

Allerdings ist aus dem vorhandenen Datenmaterial zu ersehen, dass eine wichtige Bedingung für eine eindeutige Rekonstruktion nicht gegeben ist. Bei der Banddicke überlappen die Intervalle der gesamten, je erfolgten Einstellungen von Gerüst 1 und Gerüst 2 nicht. Abbildung 4.6 zeigt die Verteilung der verschiedenen Parameter für die einzelnen Gerüste. Man erkennt, dass alle Verteilungen bis auf die Banddicke D_1 zumindest teilweise überlappen. Das heißt aber, Gerüst 1 arbeitet stets in einem anderen Arbeitspunkt als alle anderen. Und da schon ein einziger „unabhängiger“ Eingang prinzipiell ausreicht, um „entkoppelte“ Bereiche in den Wurzeln zu schaffen, ist auch nicht mit einer eindeutigen Lösung zu rechnen, solange keine weiteren Einschränkungen vorgenommen werden.

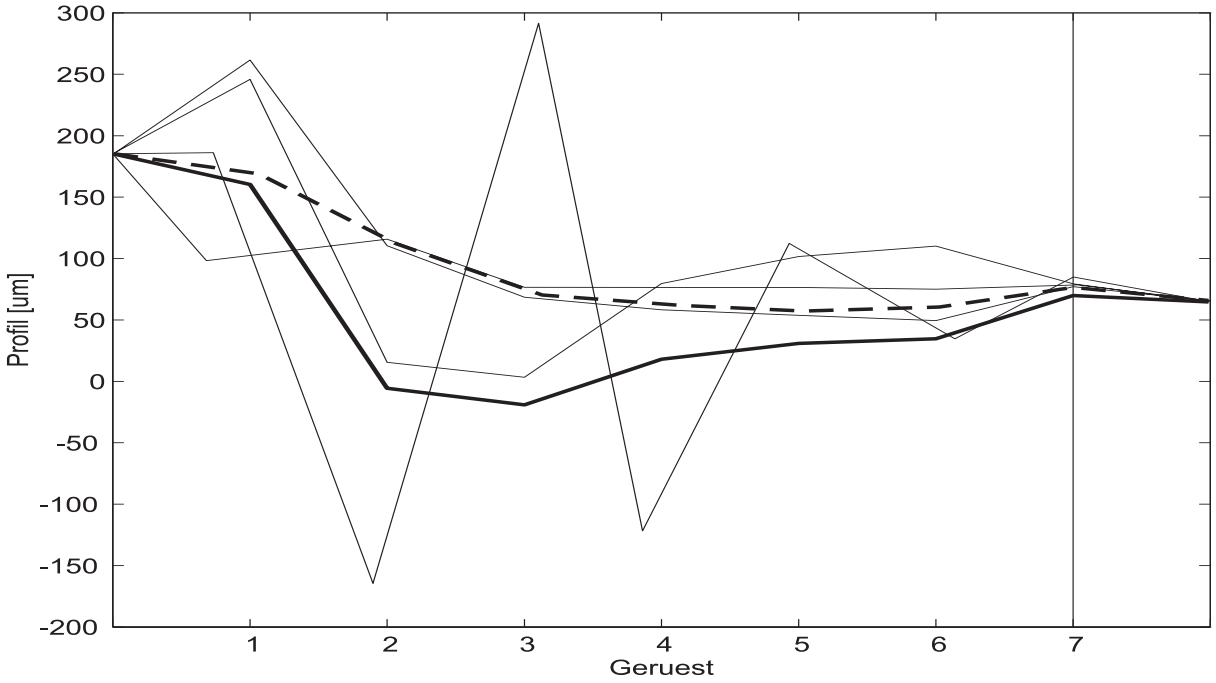

Abbildung 4.5 Zwischenprofilverläufe bei neuronaler Modellierung. Fünf verschiedene Initialisierungen führen zu fünf verschiedenen Verläufen. Gestrichelt: Profilverlauf des Standardmodells. Fett: Mit dem iterierten linearen Modell (Abschnitt 4.6) berechnete Zwischenprofile.

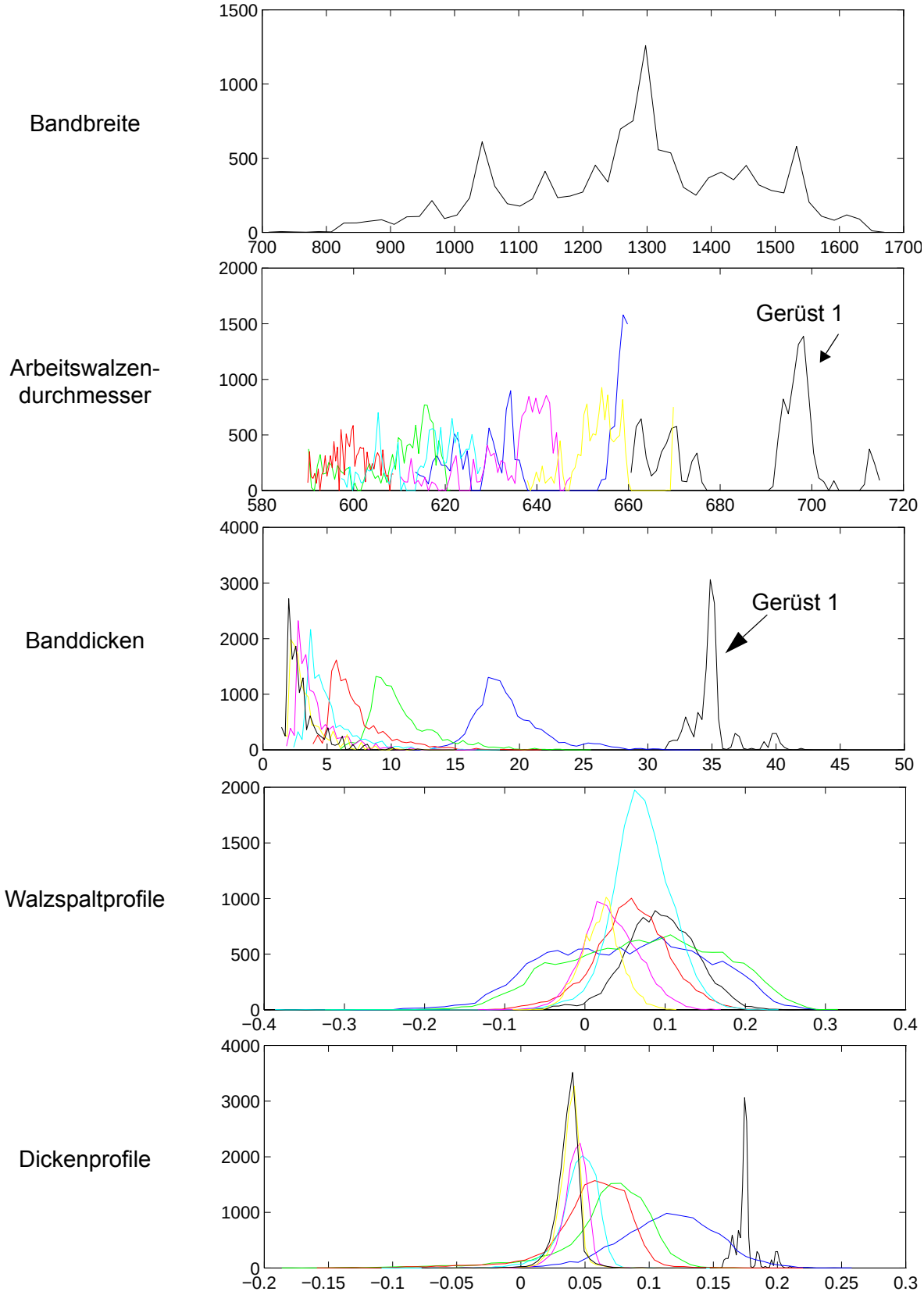

Abbildung 4.6 Histogramme relevanter Prozessgrößen. Grauwerte unterscheiden die einzelnen Gerüste. Entscheidend für die Anwendbarkeit des iterativen Netzes ist eine Überlappung jeder Verteilung mit dem Nachbarn. Dies ist bei den markierten Werten nicht der Fall.

4.6 Iteriertes lineares Modell

Ein einfaches lineares Regressionsmodell (Tabelle 4.1), das das Endprofil aus allen Mess- und Stellgrößen errechnet, liefert auf den Umstellungsbändern eine deutlich genauere Vorhersage als das zur Zeit verwendete Modell. Daher soll untersucht werden, ob eine lineare Modellierung des Einzelgerüstes zur Zwischenprofilberechnung geeignet ist.

Dabei werden alle Gerüste durch die gleiche lineare Abbildung beschrieben, d.h. das auslaufende Profil wird als Linearkombination von einlaufendem Profil und den gerüstspezifischen Parametern dargestellt:

$$p_n = \alpha p_{n-1} + a_1 W_n + a_2 B + a_3 D_n + a_4$$

Allgemein geschrieben für beliebig viele Parameter P_i, $P_1 = D, P_2 = B, P_3 = W$:

$$p_n = \alpha p_{n-1} + \sum_i a_i P_{i,n}$$

Die Koeffizienten α und a_i werden für alle Gerüste als identisch angenommen. N-faches Hintereinanderschalten dieser Gerüstübertragungsfunktion ergibt dann die Transferfunktion für das Profil über die gesamte Walzstraße:

$$p_N = \alpha\left(\ldots\alpha\left(\alpha p_1 + \sum_i a_i P_{i,1}\right) + \sum_i a_i P_{i,2}\right)\ldots\Big) + \sum_i a_i P_{i,N}$$

Zusammengefasst und nach den a_i sortiert:

$$p_N = \alpha^N p_0 + \sum_i \left(a_i \sum_{n=1}^{N} \alpha^{N-n} P_n\right)$$

Dabei bleibt diese Funktion linear in den Koeffizienten a_i. Der Koeffizient α hingegen taucht in unterschiedlichen Potenzen auf. Er kann als eine Art Decay-Faktor angesehen werden, der angibt, wie stark der Einfluss von weiter vorn in der Walzstraße liegender Gerüste auf das Endprofil abnimmt.

Dieses Modell auf einen Datensatz anzupassen bedeutet, Werte für die Koeffizienten α und a_i zu finden, die den Fehler des Endprofils minimieren. Bei gegebenem α können die optimalen a_i durch lineare Regression, z. B. mittels der Pseudoinversen, schnell bestimmt werden. Das optimale α muss durch Näherungsverfahren bestimmt werden, da es als Lösung eines Polynoms N-ten Grades für $N > 4$ nicht analytisch berechnet werden kann. Da es aber der einzige freie Parameter ist und das Intervall, in dem es liegen kann, eingeschränkt ist, können Standardverfahren zur Minimierung wie Newton eingesetzt werden. Sinnvolle Lösungen für α müssen zwischen 0 (Endprofil hängt nur vom letzten Gerüst ab) und 1 liegen.

4.6.1 Test auf Originaldaten

In Abbildung 4.7 ist die Approximationsgüte des linearen Modells in Abhängigkeit von α dargestellt. Man sieht, dass ein klares Minimum vorhanden ist. Optimiert und getestet wurde auf dem gesamten vorhandenen Datensatz von ca. 12000 Bändern. Als optimales α ergab sich ein Wert von 0.64, d.h. das einlaufende Profil geht mit diesem Faktor in das Austrittsprofil ein. Im Vergleich zum analytischen Modell ist die Genauigkeit der Endprofilvorhersage um 5% schlechter. Der Grund hierfür ist sicher auch die fehlende Online-Adaption dieses Modells. Auf den Umstellungsbändern hingegen liefert die lineare Modellierung eine gegenüber dem analytischen Modell um ~15% genauere Endprofilvorhersage. Auf den Umstellungsbändern werden dessen Adaptionskoeffizienten wieder auf Standardwerte gesetzt. Allerdings ist auch das lineare Modell auf den Umstellungsbändern deutlich schlechter als auf den Folgebändern. Das ist nur aus abweichenden Güten der Daten erklärbar, da keine Adaption stattfindet. Wahrscheinlich ist auch das aus einem Modell berechnete Walzspaltprofil auf den Umstellungsbändern nicht korrekt.

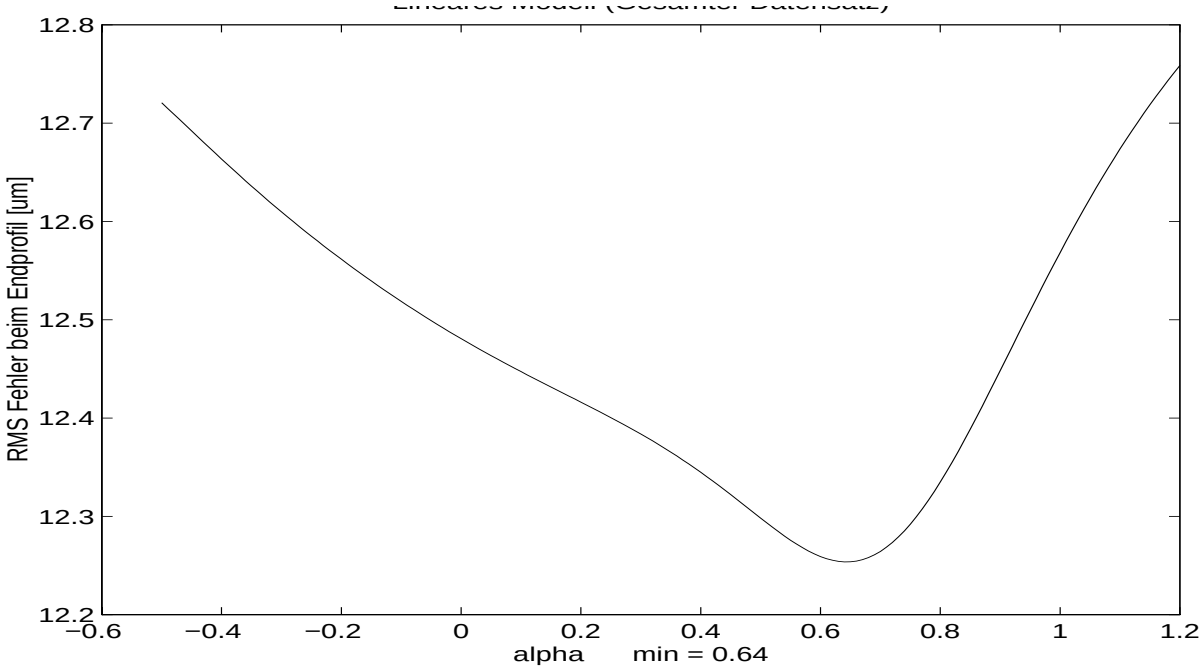

Abbildung 4.7 Einfluss des Parameters α auf die Endprofilvorhersage. Ein eindeutiges Minimum liegt bei 0.64.

Der Verlauf der Zwischenprofile unterscheidet sich deutlich von dem des analytischen Modells - der Verlauf ist regelmäßiger. Jedes nichtlineare Modell bringt z.T. extrem unterschiedliche Verläufe hervor, d.h. die Wahl des korrekten Modells ist kritisch für die Zwischenprofile. Der lineare Ansatz besitzt hohe Stabilität bei den geringsten willkürlichen Annahmen.

Methode	Alle Bänder [μm]	Umstellungsbänder [μm]
Gleitender Mittelwert	13.3	17.5
Analytisches Modell	11.8	18.7
Lineare Regression	11.9	15.1
Neuronales Netz	11.3	13.8
Gerüstweise lineares Modell	12.3	15.5
Gerüstweise lineares Modell Optimiert durch externes Netz	11.3	13.8

Tabelle 4.1 Güte der Approximation des Endprofils. Angegeben ist der RMS Fehler auf 12000 Bändern bzw. 800 Umstellungsbändern.

4.7 Hybridsystem aus linearem und neuronalem Prädiktor

Das iterierte lineare Modell liefert eindeutige Zwischenprofile, ist aber nicht in der Lage, die Physik des Walzprozesses exakt zu modellieren: Das Endprofil lässt sich mit deutlich höherer Genauigkeit vorhersagen, z.B. mit einem neuronalen Netz, das auch die Nichtlinearitäten modellieren kann. Eine Kombination beider Methoden liefert die Möglichkeit, die hohe Präzision der modellfreien Endprofilvorhersage mit der an ein Modell gebundenen Zwischenprofilberechnung zu verbinden. Dazu haben wir entsprechend Abbildung 4.8 folgendes Verfahren entwickelt:

1. Auswahl eines Datensets zur Bestimmung der Parameter des linearen Gerüstmodells, z.B. durch ein gleitendes Fenster oder besonders repräsentative Datensätze. Eventuell kann das Online-Endprofilmodell die Auswahl vornehmen.
2. Anpassung der Koeffizienten des linearen Modells anhand dieser Daten.
3. Vorhersage des Endprofils des aktuellen Bandes mit diesem Modell.
4. Vorhersage des Endprofils des aktuellen Bandes mit einem neuronalen Modell.
5. Vergleich der Ergebnisse und Korrektur der Koeffizienten des linearen Gerüstmodells, so dass das gleiche Endprofil herauskommt. Diese Änderung gilt nur für das aktuelle Band.

Die Änderung der Koeffizienten erfolgt so, dass die Summe aller Änderungen von a_i und α so klein wie möglich ist. Damit bleibt das lineare Gerüstmodell trotz der Korrektur dem ursprünglichen, optimal angepassten Modell so ähnlich wie möglich. Das ist auf zwei Weisen erreichbar: Bei normierten Daten sollte die Änderung aller Koeffizienten gleich groß sein, d.h. der Euklidische Abstand zwischen altem und neuem Koeffizientenvektor sollte minimal sein. Als Normierungsverfahren sollten dann alle Inputs so skaliert werden, dass alle Koeffizienten des linearen Modells zu „1“ werden. Bei nicht normierten Inputs sollten alle Koeffizienten mit einem möglichst ähnlichen Faktor

multipliziert werden. Getestet wurde auch ein Gradientenverfahren, das alle Koeffizienten langsam verändert bis das gewünschte Ergebnis am Ausgang ansteht.

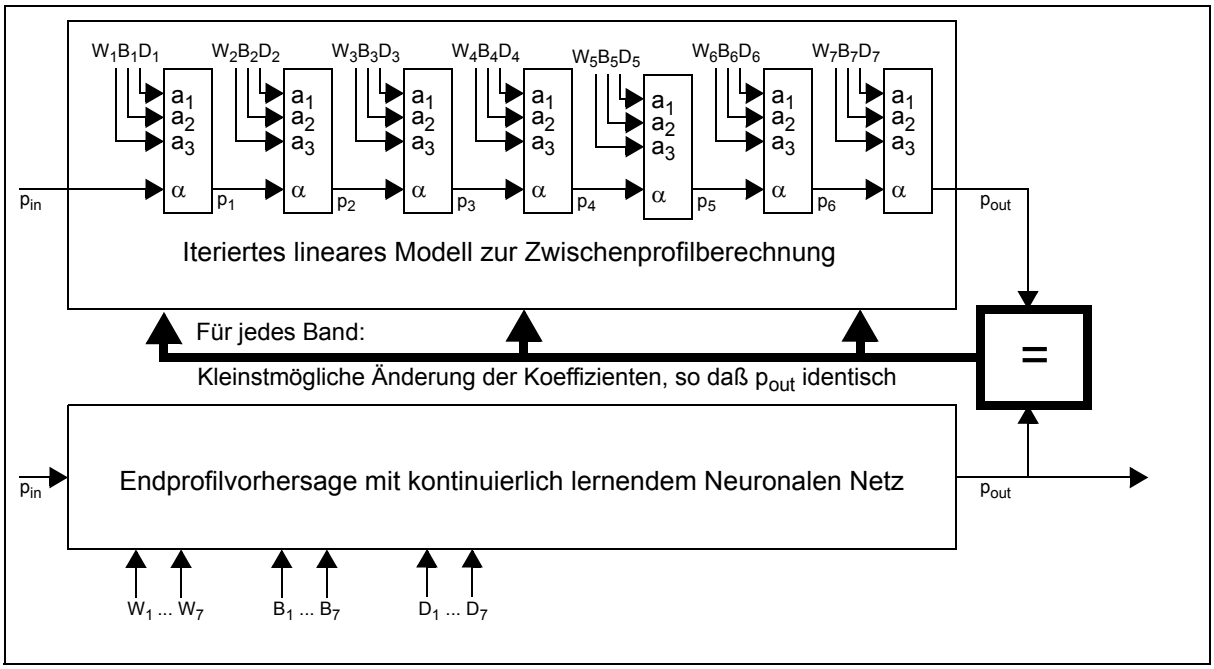

Abbildung 4.8 Hybrides Modell zur Profilvorhersage bei einer Walzstraße. Das Gerüstmodell berechnet aus einlaufendem Profil p_{in} und den Gerüsteinstellungen W, B, D die Zwischenprofile p_n und das Endprofil p_{out}. Das neuronale Netz berechnet ebenfalls das Endprofil p_{out}, allerdings ist das Ergebnis in der Regel genauer. Daher werden die Parameter a, b, c des Gerüstmodells so adjustiert, dass es das gleiche Endprofil wie das NN erzeugt.

Die Anwendbarkeit einer solchen Kombination ist nicht auf ein rein lineares Modell zur Zwischenprofilvorhersage beschränkt. Auch bei anderen, die Physik des Walzprozesses besser beschreibenden mathematische Gleichungen könnte man deren freie Parameter zunächst global optimieren und dann für jedes Band so modifizieren, dass sich das vom neuronalen Modell prognostizierte Endprofil ergibt.

Abb. 4.9 zeigt zum Vergleich die Profilverläufe von analytischem Modell und Hybridmodell, jeweils auf ca. 800 Umstellungsbändern einer Anlage. In Tabelle 4.1 ist die Güte der Approximation angegeben. Zum Training wurden alle vorhandenen Bänder vor dem aktuellen und alle ab 1000 Bändern nach dem aktuellen Band verwendet. Der Datensatz bestand aus ca. 12000 Bändern, davon 800 Umstellungsbänder.

Diese Form der Parallelschaltung von Modellen hat Vorteile insbesondere bezüglich der Zuverlässigkeit und Sicherheit der Verfahren im industriellen Einsatz. Das mathematische Modell ist robust und kann bei Bedarf ohne die neuronale Optimierung verwendet werden, auch wenn dies zu suboptimalen Ergebnissen führt. Erst wenn das Netz durch ausreichendes Training verlässlich bessere Werte liefert, wird sein Ausgangswert zur Optimierung der Parameter des mathematischen Modells herangezogen.

Auf dieses Verfahren wurde ein Patent erteilt [Kindermann et.al. 1999] und befindet sich zur Zeit in der industriellen Erprobung.

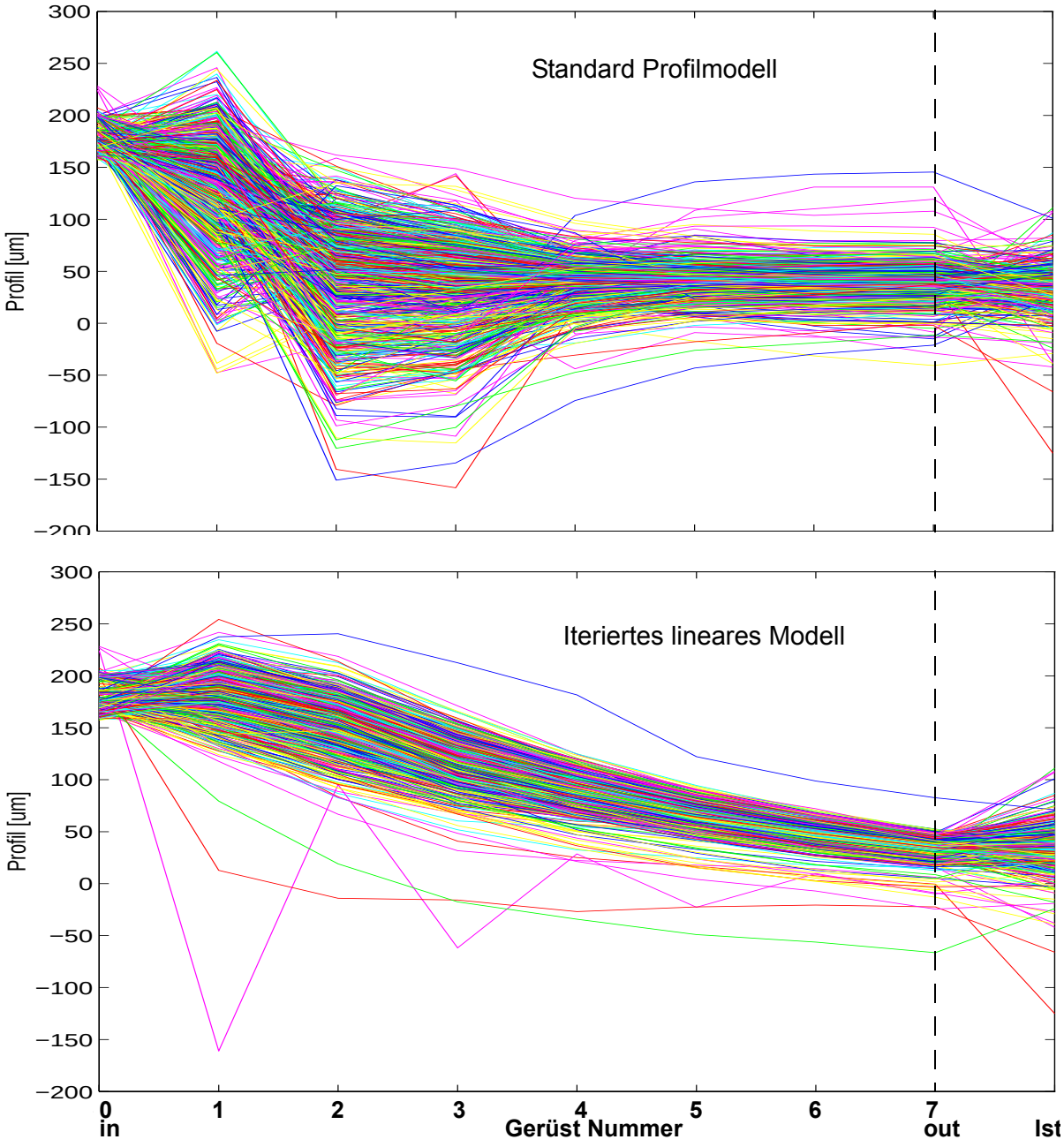

Abbildung 4.9 Vergleich der Zwischenprofilverläufe. Oben das Standardmodell und unten das lineare Modell. Ganz links jeweils das Eintrittsprofil, dann die berechneten sechs Zwischenprofile und das Endprofil. Zum Vergleich rechts das tatsächlich gemessene Endprofil.

KAPITEL 5

Ausblick

5.1 Einordnung in die aktuelle Entwicklung

Der überwiegende Teil der Literatur auf dem Gebiet der iterativen Wurzeln (und ähnlicher Funktionalgleichungen) befasst sich auf sehr abstraktem mathematischem Niveau mit den Eigenschaften dieser Objekte. Für jemanden, der sich mit einem praktischen Problem konfrontiert sieht, das die konkrete Lösung einer iterativen Wurzel erfordert, war bis vor kurzem so gut wie keine Hilfe verfügbar. Erst in jüngster Zeit wird verstärkt über Methoden nachgedacht, die anwendungsbezogener sind.

Castillo und Iglesias [1997] sind dabei, ein symbolisches Paket für Mathematica zu entwickeln, das erlauben soll, symbolische Lösungen für Funktionalgleichungen zu finden. Sie führen folgende Vermutungen an, warum etwas Vergleichbares bisher nicht verfügbar ist:

- „Es ist ein schwieriges Problem. Bei der Implementation dieses Typs von Gleichungen müssen verschiedene Faktoren bedacht werden:

  Die Lösung einer Funktionalgleichung hängt stark vom Definitionsbereich und der Klasse der zulässigen Funktionen ab.

  Beliebige Konstanten und beliebige Funktionen können in der allgemeinen Lösung von Funktionalgleichungen auftauchen.

  Anders als alle anderen Arten von Gleichungen, kann eine einzelne Gleichung mehrere unbekannte Funktionen festlegen.

- Es besteht wenig Bedarf. Funktionalgleichungen sind kaum bekannt und werden selten angewandt. Es ist üblich andere Methoden zu verwenden, sogar wenn Funktionalgleichungen die beste Option zu einem Problem wären.

- Es gibt keine klare Methodologie, Funktionalgleichungen zu lösen. Im Gegenteil, in vielen Fällen sind „ad hoc“ Lösungen nötig. Dies bedeutet eine große Schwierigkeit, Algorithmen zu implementieren.“

Ihr Ansatz geht in die Richtung, eine Bibliothek aller bekannten lösbaren Funktionalgleichungen aufzubauen und die Mustererkennungsfähigkeiten von Mathematica dazu zu verwenden, eine eingegebene Funktionalgleichung mit dieser Sammlung zu vergleichen und ggf. die Lösung zurückzugeben.

Daneben gibt es einige wenige, mehr approximative Ansätze, die eher an numerischen Lösungen interessiert sind:

Einen Versuch, über formale Potenzreihen zu Lösungen von iterativen Wurzeln zu gelangen, macht Briggs [1995]. Die Schwierigkeiten sind jedoch nicht unerheblich. Es treten „Multinome“ auf, die nicht mehr als Reihe, sondern nur noch als Matrix dargestellt werden können und Konvergenzradien sind sehr beschränkt. Allerdings werden diese Schwierigkeiten vielleicht einleuchtend, wenn man bedenkt, dass iterative Wurzeln ja selbst auf den Polynomen noch nicht befriedigend gelöst sind [Baron 2001].

Ähnlich stellen Aldrovandi und Freitas [1998] Funktionen als formale Potenzreihen von Bell-Polynomen dar, die als Bell-Matrizen dargestellt werden können [Comtet 1974]. Die Potenzen dieser Matrix nach Faá di Bruno [1855] werden dann wieder als Polynome dargestellt und als kontinuierliche Iteration der Funktion interpretiert. Sie können damit fraktionale Iterationen der logistischen Abbildung beschreiben und zeigen eine Anwendung für die Modellierung turbulenter Strömungen.

Kozba [2000] gibt Methoden an, wie iterative Quadratwurzeln von stückweise linearen monotonen Funktionen, also Polygonzügen, numerisch angenähert werden können. Es zeigt sich, dass schon der Übergang von einer „normalen“ linearen Funktion zu einer stückweise linearen nicht unerhebliche Schwierigkeiten mit sich bringt: Selbst wenn ein Polygonzug kontinuierliche Wurzeln besitzt, müssen diese selbst nicht unbedingt als Polygonzug darstellbar sein.

Alle diese Methoden basieren jedoch auf gegebenen Funktionen, deren Gleichung bekannt ist. Der hier neu vorgestellte Einsatz neuronaler Netze zur Lösung kann als weiterer Baustein für die Nutzbarmachung der verallgemeinerten Iteration gewertet werden: Wenn eine Funktion, deren Wurzeln berechnet werden sollen, gar nicht explizit gegeben ist, sondern nur in Form von Datensätzen, stellt sie eine darauf direkt anwendbare numerische Methode dar. Gerade in Fällen, bei denen neuronale Netze sowieso schon zur Systemidentifikation eingesetzt werden, ist diese Methode optimal geeignet.

5.2 Übertragung auf weitere Funktionalgleichungen

Iterative Wurzeln stellen die Lösung der Funktionalgleichung $\varphi^n = f$ dar. Daher liegt die Frage nahe, ob auch andere Funktionalgleichungen mit ähnlichen Methoden durch neuronale Netze darstellbar und lösbar sind.

Viele Phänomene lassen sich sehr intuitiv als Funktionalgleichung formulieren und lösen [Aczél & Dhombres1989]. Diese Methoden werden trotz der formalen Eleganz bisher jedoch kaum angewandt, vermutlich wegen der mathematischen Probleme und weil diese Möglichkeiten vielen Anwendern gar nicht bekannt sind. Andererseits benutzt man oft Funktionalgleichungen, um Zusammenhänge zum Ausdruck zu bringen und später, nachdem man mit anderen Verfahren eine Lösung gefunden hat, zu zeigen, dass diese Lösung tatsächlich der Funktionalgleichung entspricht. Die Funktionalgleichung direkt anzugehen, wird gar nicht in Erwägung gezogen.

Hier sollen in knapper Form einige Ideen skizziert werden, wie Funktionalgleichungen durch ein neuronales Netz repräsentiert werden können und damit evtl. numerische Lösungen bestimmt werden können.

5.2.1 Die Inverse

Die Gleichung $\varphi(f(x)) = x$ wird von der Inversen $\varphi = f^{-1}$ gelöst. Die Inverse einer Funktion mit einem neuronalen Netz zu bestimmen, ist keine Schwierigkeit. Die einfachste Möglichkeit besteht darin, in den Trainingsdaten Eingänge und Ausgänge zu vertauschen. Aber auch Methoden, die Gewichtsmatrix eines fertig trainierten Netzes gewissermassen zu „invertieren“, sind verfügbar [Linden & Kindermann 1989]. Dies kann als Basis für weitere Gleichungen verwendet werden.

5.2.2 Periodizität

Die Bedingung, dass eine Funktion periodisch ist, $\varphi(x) = \varphi(x + c)$, lässt sich durch nochmaliges Präsentieren der Trainingsdaten unter Addition von c zu den Eingangsdaten der Kopie ausdrücken.

5.2.3 Schröder Gleichung

Die Schröder Gleichung [1871]

$$(\varphi(f(x)) = c\varphi(x)),\ c \in \mathbb{R}$$

ist eine der grundlegendsten Funktionalgleichungen. Sie repräsentiert das Eigenwertproblem der Funktionalalgebra [Kuczma 1973].

Für bijektive φ kann sie umgeformt werden zu $f(x) = \varphi^{-1}(c\varphi(x))$. Diese Gleichung kann dann auf das Netzwerk in Abbildung 5.1 abgebildet werden. Nur die Gewichte des zweiten Teilnetzes werden herkömmlich trainiert. Gleichzeitig wird es kontinuier-

lich invertiert und auf das erste Teilnetz abgebildet. Die Konstante aus der Gleichung wird im Netz durch ein festes Gewicht zwischen den Teilnetzen repräsentiert. Konvergiert die Kombination aus Training und gleichzeitiger Inversion, entspricht das erste Teilnetz der gesuchten Lösung φ .

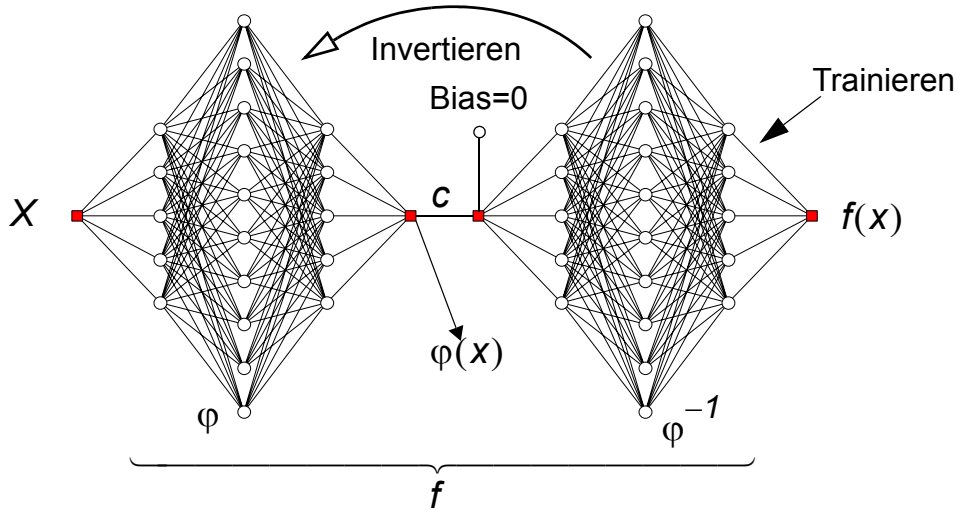

Abbildung 5.1 Eine neuronale Repräsentation der Schröder Gleichung.

5.2.4 Die Abel Gleichung

Die Abel'sche Funktionalgleichung

$$\varphi(f(x)) = \varphi(x) + c$$

kann durch ein fast identisches Netz dargestellt werden. Der ebenfalls gesuchte Wert c ist hier allerdings durch den Bias des ersten Neurons des zweiten Teilnetzes ablesbar. Das Gewicht zwischen den Teilnetzen ist konstant 1.

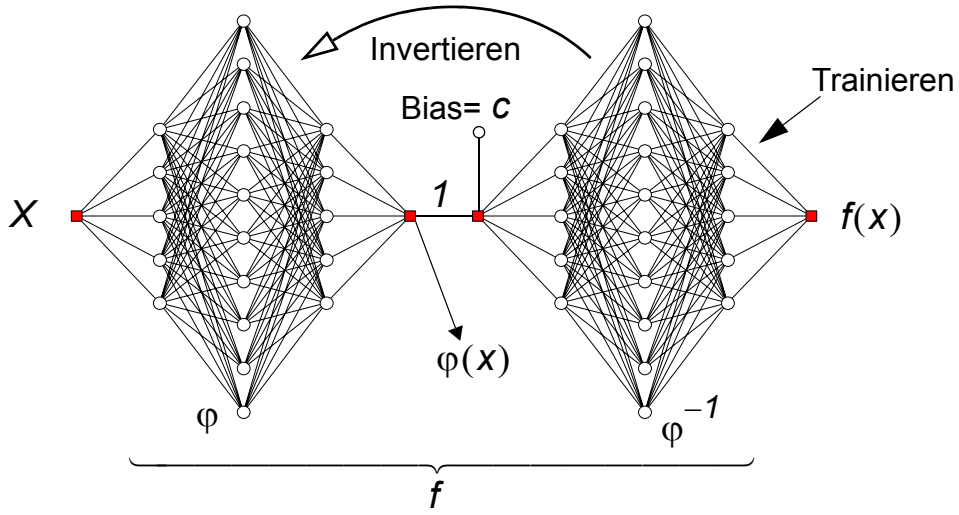

Abbildung 5.2 Netz für die Abel-Gleichung. Einziger Unterschied zu Abbildung 5.1 ist der Ort der "Einspeisung" von c.

5.2.5 Kommutierende Funktionen

Lösungen φ der Gleichung

$$\varphi(f(x)) = f(\varphi(x))$$

sind alle kommutierende Funktionen von f. Abbildung 5.3 zeigt ein Netz, das f lernt und zusätzlich eine damit kommutierende Funktion φ findet. Wenn die Lösungsmenge sehr groß ist, macht dies allein wenig Sinn, jedoch könnten weitere Netze zusätzliche Bedingungen für φ vorgeben.

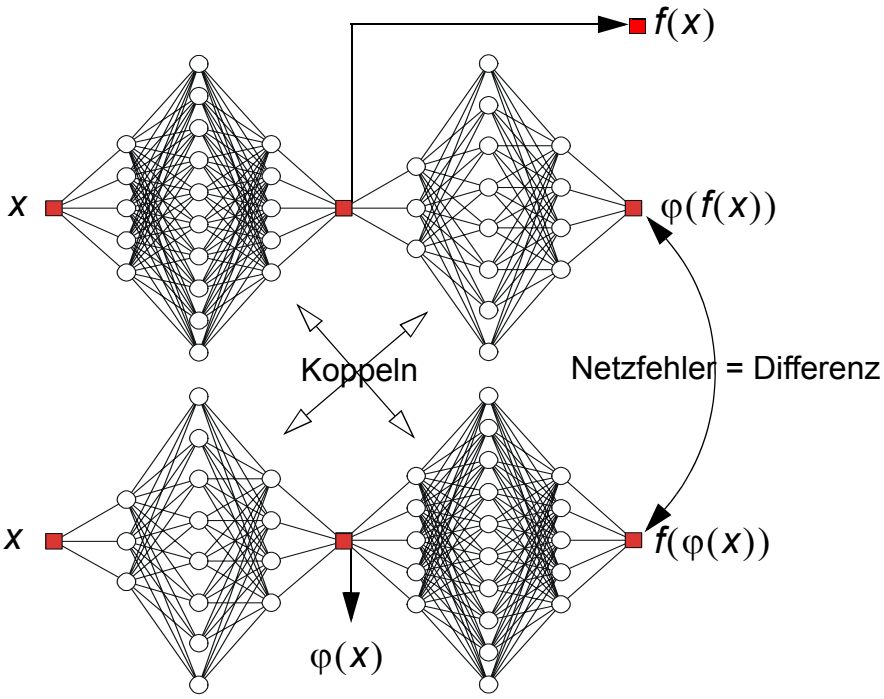

Abbildung 5.3 Netz für kommutierende Funktionen.

5.2.6 Feigenbaum Gleichung

$$c\varphi(x) = \varphi(\varphi(cx))$$

Diese Gleichung (Abb 5.4) spielt eine wichtige Rolle in der Theorie der quadratischen Abbildungen und in der Renormalisierungstheorie dynamischer Systeme [Brigs et. al.1988].

5.2.7 Systeme von Funktionalgleichungen

Die hier gezeigten Beispiele sollten eine Idee vermitteln, wie Funktionalgleichungen in neuronale Netze übersetzt werden können. Oft wird aber ein Problem durch ein System aus mehreren Funktionalgleichungen beschrieben. Enthalten alle Gleichungen eine gemeinsame Funktion f, können mehrere zunächst unabhängige Netze, die

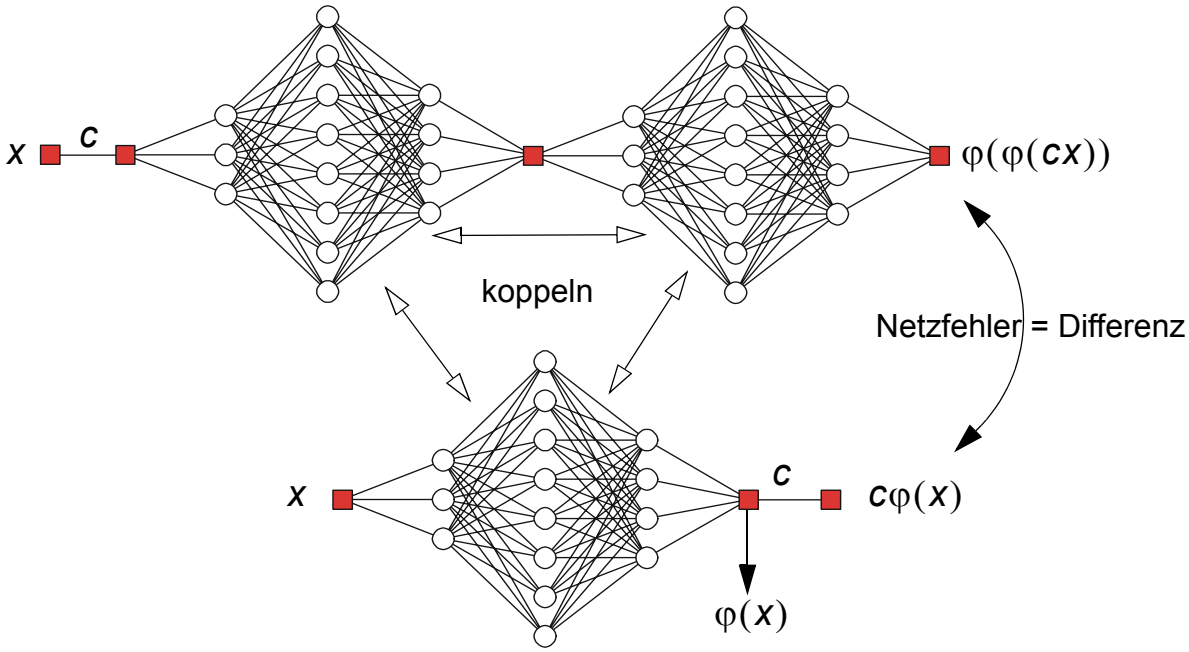

Abbildung 5.4 Netz, dass die Feigenbaum-Gleichung repräsentiert.

jedes diese Funktion als Subnetz enthalten, zu einer Repräsentation dieses Gleichungssystems verknüpft werden. Dabei werden die Netze durch Kopplung des *f*-Teilnetzes miteinander verknüpft.

Zum Schluss noch ein Gedanke, der die Umkehrung dieser Ideen darstellt. Es wurde gezeigt, dass definierte neuronale Strukturen im Gehirn Anteil an ganz unterschiedlichen Prozessen haben. Einzelne kortikale Kolumnen können funktional zu mehreren Hyperkolumnen gehören und an den Prozessen von beiden aktiv teilhaben. Dies erinnert an Funktionen, die in mehreren Funktionalgleichungen vorkommen und diese dadurch koppeln. Vielleicht kann eine abstrakte Beschreibung solcher Hirnstrukturen als System gekoppelter Funktionalgleichungen einen Ansatz zur Erklärung mancher neuronaler Zusammenhänge liefern.

ANHANG A Implementierungen

A.1 FORWISS Artificial Neural Network Simulation Toolbox

FAST ist ein im Bayerischen Forschungszentrum für Wissenbasierte Systeme (FORWISS) entwickeltes Programm zur Simulation von Neuronalen Netzen, das über das Internet frei verfügbar ist. Es ist kommandozeilenorientiert und auf hohe Geschwindigkeit optimiert. Es gibt Versionen für Dos, Windows und alle gängigen Unix und Linux Versionen [http://www.forwiss.uni-erlangen.de/aknn]. Lediglich zwei neue Parameter, *nIter* und *betaShare* werden in den Netzdefinitionen benötigt, um ein gegebenes Netz mehrfach hintereinanderzuschalten und über „Weight Sharing“ eine Angleichung der Gewichte zu forcieren. Siehe „Koppeln der Gewichte“ auf Seite 54.

- *nIter* gibt an, wie viele Kopien eines Netzes hintereinandergeschaltet werden sollen, entsprechend der Modellierung der n-ten iterativen Wurzel.
- *betaShare* gibt den Kopplungsfaktor zwischen korrespondierenden Gewichten der Teilnetze entsprechend Gleichung Gleichung (3.4) an. Standardeinstellung ist 0.5

Im folgenden Beispiel wird die Anwendung zur Berechnung der iterativen Wurzel von $f(x) = x^2 + 1$ demonstriert. Das File data.dat enthält eine Tabelle von 20 Funktionswerten im ASCII Format.

```
AnnType MLP
nInputs 1
nHidden 5
nOutputs 1
OutputType Linear
nIter 2
BetaShare 0
WeightFileOut weights.dat
Train data.dat
```

```
AnnType MLP
nInputs 1
nHidden 5
nOutputs 1
OutputType Linear
nIter 2
BetaShare 0.5
WeightFileIn weights.dat
Train data.dat
```

Tabelle A.1 Kontrollfiles für FAST. Links Vortraining, rechts mit Koppeln der Gewichte.

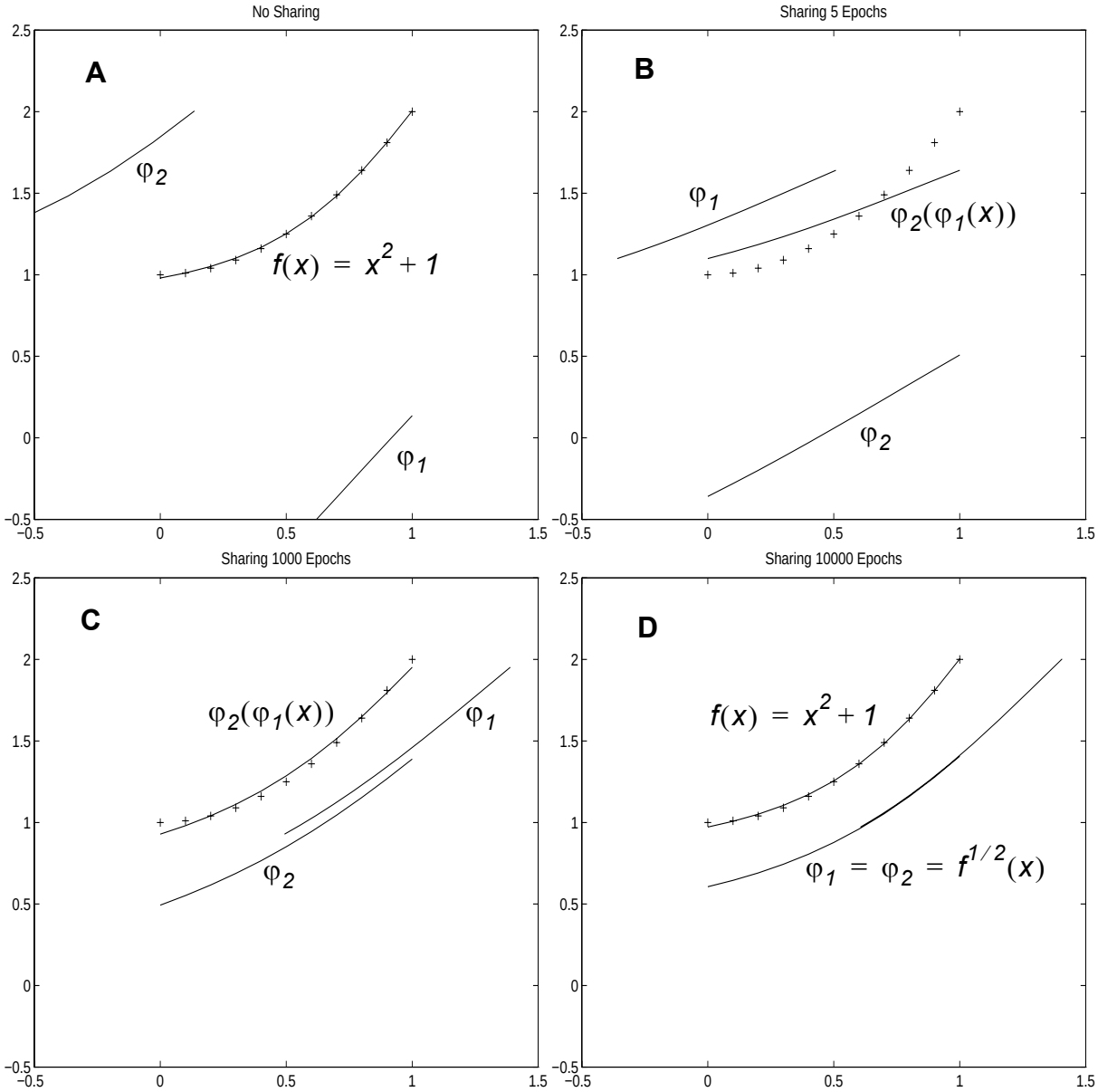

Abbildung A.1 Darstellung des Lernvorganges mit FAST. A: Die Funktion $f(x) = x^2 + 1$ wurde von einem normalen (1-5-1-5-1) Netz gelernt. Die Teilnetze haben völlig unterschiedliche Funktionsverläufe. B: Beim Aktivieren der Gewichtskopplung nähern sich φ_1 und φ_2 einander an, die Fähigkeit zur Modellierung von f geht jedoch zunächst verloren. CD: Erst längeres Trainieren führt zur Approximation von f und zu identischen φ, die jetzt der iterativen Wurzel von f entsprechen.

A.2 ECANSE Worksheet

ECANSE ist ein integriertes Entwicklungssystem der SIEMENS AG, das unter einer grafischen Oberfläche datenflussorientierte Programmierung erlaubt. Eines der Haupteinsatzgebiete ist die Netzlastprognose für die Stromindustrie. [Böhm 1995]. Es

dient auch als Programmieroberfläche für den ersten kommerziellen Hardware Neurocomputer, die Synapse [Anlauf 1995].

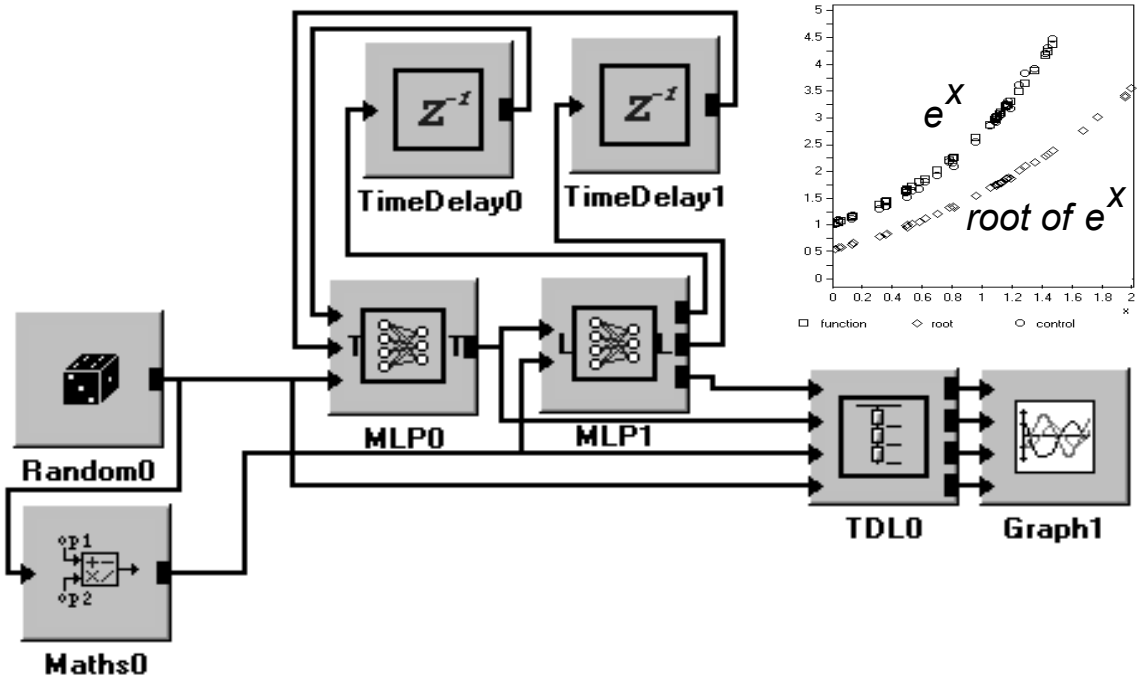

Abbildung A.2 Implementation von iterativen Wurzeln in ECANSE.

Abbildung A.2 zeigt als Beispiel, wie online die Funktion e^x und ihre iterative Wurzel gelernt werden. Das Random-Objekt erzeugt die x Werte als Zufallszahlen, das Maths Objekt berechnet daraus einen Funktionswert $f(x)$, dort kann eine beliebige Funktion eingetragen werden. Alternativ könnten externe Daten aus einer Datei oder direkt von Messdatenaufnehmern eingespeist werden.

Der x-Wert wird durch MLP0 geschickt. Das Ergebnis wird als Eingang in MLP1 gespeist, dazu $f(x)$ als Zielvorgabe. MLP1 lernt aktiv diesen Zusammenhang, gleichzeitig werden die Werte aller Gewichte permanet auf MLP0 übertragen (Ein TimeDelayObjekt für jede Schicht), beide Netze sind also identisch. Die Toplogie ist 1 Eingang, 1 Ausgang und 5 Neuronen in der versteckten Schicht

Mit dem Graph-Objekt werden ständig $f(x)$ (die Trainingsdaten) sowie die Netzausgabe von MLP1 (die angepasste Funktion) und der Ausgang von MLP0 (die iterative Wurzel von f) dargestellt.

Bei Standardeinstellungen braucht Ecanse ca. 3000 Epochen, eine solche Kennlinie zu lernen (MLP1 alleine), das Hinzufügen von MLP0 zur gleichzeitigen Berechnung der Wurzel verdreifacht ungefähr die nötigen Lernschritte.

Diese relativ einfache Implementation erlaubt es Anwendern, ohne eigenen Programmieraufwand iterative Wurzeln zu modellieren.

A.3 MATLAB Neural Network Toolbox

Das Programm *MATLAB* der Firma Mathworks (www.mathworks.com) ist ein auf der LINPACK-Bibliothek basierendes System zur numerischen Manipulation von Daten. Es hat sich im naturwissenschaftlichen und technischen Bereich zu einer Standardsoftware entwickelt.

Zahlreiche sogenannte Toolboxen erweitern den Funktionsumfang, darunter die Neural Network Toolbox, mit der auf einfache Weise neuronale Algorithmen in Applikationen integriert werden können.

Ein Teil der in dieser Arbeit dargestellten Methoden zur numerischen Berechnung iterativer Wurzeln wurde in MATLAB als Erweiterung dieser Toolbox erstellt. Dabei wurde Wert auf weitestgehende Kompatibilität zu den vorhandenen Funktionen gelegt. Funktionen, die mit „iterativen“ Netzen operieren, erhalten den gleichen Namen wie ihr Original mit angefügter Endung „it“.

Damit sollte für Anwender der Neural Network Toolbox die Verwendung dieser Funktionen so einfach und intuitiv wie möglich sein.

A.3.1 newffit

Basiert auf der MATLAB Funktion *newff* (new feed forward network). *newff* erzeugt ein Netzwerk mit einer Topologie, definiert durch die Zahl der Eingänge, Ausgänge und die Größe der verdeckten Schicht bzw. Schichten. *newffit* fügt einen einzigen weiteren Parameter hinzu, die Anzahl der Iterationen, die dieses Netz durchläuft, bzw. die Anzahl der iLayer. *newffit* konstruiert daraus ein entsprechend großes „normales“ Netz, setzt einige neue Parameter und stellt die Fehlerfunktion für das Training auf *mseit* ein.

newff(...) $\rightarrow$ *newffit (N,...)*

N bedeutet die Anzahl der hintereinanderzuschaltenden Teilnetze. Alle anderen Parameter sind identisch zur Standardfunktion *newff.*

A.3.2 mseit

Basiert auf der MATLAB funktion *mse* (mean square error), die den mittleren quadratischen Fehler auf dem Trainingsdatensatz am Netzausgang berechnet. In *mseit* wird zusätzlich die Varianz der Gewichte über die iLayer berechnet (siehe Abschnitt 3.4.3) und mit dem Approximationsfehler gewichtet addiert, was zu einer Regularisierung des Netzes führt. Für Kopplungsstärke 0 bzw. Netze mit nur einer iLayer liefert *mseit* identische Werte wie *mse*. (Vergleiche dazu die Standardfunktion *msereg.*)
Diese Funktion wird in der Regel nicht vom Benutzer selbst aufgerufen, sondern definiert die Fehlerfunktion für das Netztraining.

A.3.3 dmseit

Basiert auf der MATLAB funktion *dmse* (derivative of mean square error). Der Backpropagation Algorithmus benötigt die partiellen Ableitungen der Fehlerfunktion nach den einzelnen Gewichten. Daher definiert MATLAB zu jeder Fehlerfunktion eine entsprechende Routine, die diese Ableitungen berechnet. *dmseit* stellt diese Funktionalität für *mseit* zur Verfügung und ist ebenfalls als interner Bestandteil anzusehen.

A.3.4 getxit

Erweiterung der MATLAB Funktion *getx*. *getx* ist eine low-level Funktion, die alle Gewichte einschließlich der Bias-Gewichte eines Netzes in einem einzigen Vektor zusammenfasst. *getxit* liefert ebenfalls alle Gewichte eines Netzes, jedoch als eine Matrix, deren Spalten jeweils alle Gewichte eines iLayers enthalten. Für ein perfekt trainiertes iteratives Netz sollten alle Spalten praktisch gleich sein. Für Netze mit nur einem iLayer liefert sie das gleiche Ergebnis wie *getx*. *getxit(net,i)* liefert nur die Gewichtsmatrix des i-ten Teilnetzes.

A.3.5 setxit

Erweiterung der MATLAB Funktion *setx*. *setx* ist das Pendant zu *getx* und erlaubt alle Gewichte eines Netzes in einem Vektor auf einmal zu setzen. *setxit* erlaubt dagegen eine Matrix als Argument wie sie die Funktion *getxit* liefert.

A.3.6 train

Die normale Matlab Funktion *train* kann unverändert auch für iterierte Netze verwendet werden!

A.3.7 sim

Auch *sim* muß nicht verändert werden. Wird sie auf ein iteriertes Netz angewandt, ist das Ergebnis eine Matrix, deren Spalten nun auch die Netzausgaben der Teilnetze enthalten. Die letzten Spalten sind die übliche Ausgabe des Netzes am „Ende“.

A.3.8 pretrainit

Diese Funktion dient zum optionalen Vortrainieren eines „iterativen Netzwerks“ (siehe Abschnitt 3.7). Damit sind im Wesentlichen zwei Vorteile verbunden: Eine gegebene Funktion $f(x)$ hat oft mehrere, häufig stark voneinander abweichende Lösungen. Beispielsweise hat die Funktion $f(x) = 2x$ u.a. die Funktionswurzeln $\varphi_1(x) = \sqrt{2}x$ und $\varphi_2(x) = a - \sqrt{2}x$, a beliebig. Initialisiert man das Netz mit zufälligen Gewichten, wird beim Training ebenfalls zufällig eine dieser Lösungen angesteuert. Ist man aber z.B. nur an einer Lösung mit positiver Steigung interessiert, kann man das Netz in diese

Richtung vorprägen. Dadurch wird die Wahrscheinlichkeit drastisch erhöht, bei *train* eine Lösung der gewünschten Form zu erhalten.

[net guess] = pretrainit(net,inputs,targets,typ)

guess gibt optional das Ziel des Vortrainings an, je nach dem Wert von *typ*.

typ kann die Werte 0 bis 3 annehmen: (Standard = 1)

0: kein Vortraining

1: Training auf die Identität y = x

2: Training auf die Funktion

3: Training auf Mittelwert zwischen Funktion und y = x

A.3.9 meanit

Nach *train* können die Subnetze noch voneinander abweichen. *meanit* erzeugt durch Mittelwertbildung der Gewichte absolut identische Subnetze.

net = meanit (net)

A.3.10 plotnet

Eine allgemein verwendbare - in MATLAB oft vermisste - Funktion, ein neuronales Netz zu zeichnen. Sie kann auch für nicht-iterierte Netze verwendet werden.

plotnet (net)

A.3.11 Network Objekt

MATLAB definiert ein *network* Objekt, das ein komplettes Netz repräsentiert, mit Ein-Ausgangsdefinition, Verbindungsstruktur, Aktivierungsfunktionen sowie Voreinstellungen für Fehlerfunktion und Lernverfahren. Um iterierte Netze zu kennzeichnen und den „it“ Funktionen Details über die Netzstruktur mitzuteilen, werden zusätzlich die folgenden Felder definiert:

network.userdata.note. Enthält stets den String “Iterative Root Network“ zur Information

network.userdata.IT. Anzahl der Iterationen bzw. die Ordnung der Wurzel, die jedes Teilnetz modellieren soll. IT=1 bedeutet ein normales, nicht iteriertes Netz, .

network.userdata.DIM. Anzahl der Ein- und Ausgänge, beide sind bei iterierten Netzen identisch.

network.userdata.PR. Layerstruktur des Teilnetzes, wie in *newffit* übergeben.

network.performParam.ratio. Dieser Wert stellt ein, wie in der Fehlerfunktion Approximationsfehler und Regularisierungsfehler gewichtet werden sollen. Ein Wert von Eins deaktiviert die Regularisierung, das Netz wird wie ein normales Netz trainiert, die Subnetze werden i.A. völlig unterschiedlich aussehen. Ein Wert nahe Null stellt die Regularisierung in den Vordergrund, ggf. auch auf Kosten der Genauigkeit der Approximation der Targetfunktion. Standardwert ist 0.5.

A.3.12 Beispiel: Die iterative Wurzel der e-Funktion

Dieses Beispielscript demonstriert, wie in Matlab die iterativen Wurzeln der Exponentialfunktion berechnet werden können:

```
% Create a Function Table and Plot the Graph
x=[-2:0.1:1] %x-range
t=exp(x)     %function
plot(x,t)
title('f(x)=e^x')

% Create an iLayer'ed Neural Network and Plot the Structure
net=newffit(4,[min(x),max(x)],[8 3])
plotnet(net)
```

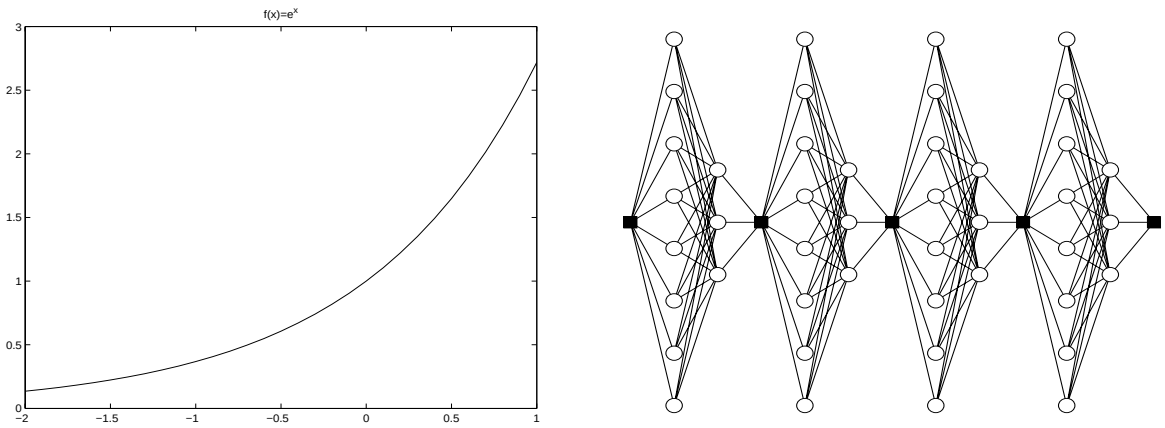

Abbildung A.3 Die Funktion e^x und ein 4-fach gestaffeltes Netzwerk.

```
% Pretrain each iLayer towards a guessed Solution (optional)
net.trainParam.goal = 1E-4 % low accuracy only
net=pretrainit(net,x,t,3)  % 3=try a solution between f and y=x

% Display the Result of this Procedure (optional)
y=sim(net,x);
plot(x,t,'o', x,y(1,:),'k', x,y(end,:),'r.-', x,x,'k:')
legend('target','guess','output','y=x')
title('pretraining to e^x')
```

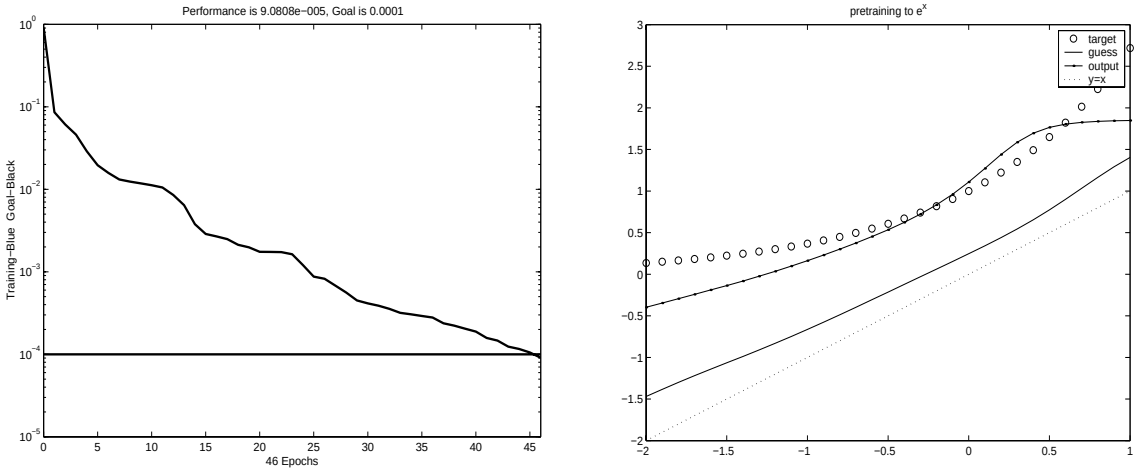

Abbildung A.4 Lernkurve und Ergebnis des Vortrainings. Die Genauigkeit muss nur gering sein. Man sieht, dass die Iteration der „vermuteten“ Wurzel nicht sehr überzeugend ausfällt.

```
% Setup learning Parameters (optional)
%net.trainParam.goal    = 1E-8  % min desired Error
%net.trainParam.epochs  = 500   % max Epochs
%net.performParam.ratio = 0.5   % 1:Only Error 0:Only Weights


% Train and Evaluate the Net and show the Result
net=train(net,x,t)
y=sim(net,x)
plot(x,t,'ko', x,y,'k', x,y(end,:),'r.-', x,x,'k:')
legend('target','4-th root','root','f^{3/4}','','output','y=x')
title('fractional iterations of e^x')
```

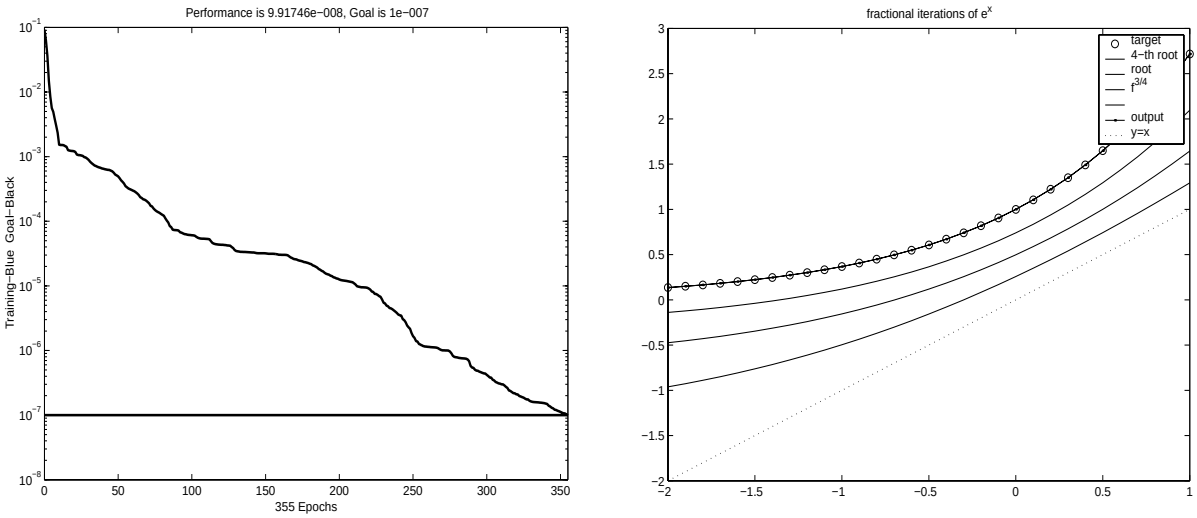

Abbildung A.5 Lernkurve und Ergebnis der Trainings. Das Endergebnis stellt hier sehr schön die 4-te iterative Wurzel von $f(x) = e^x$ sowie deren Iterationen, also $f^{1/2}$ und $f^{3/4}$ dar.

A.3.13 Listings der Funktionen

Dank der MATLAB Toolbox, die einen sehr hohen Abstraktionsgrad ermöglicht, sind die meisten Funktionen kurz genug, um hier komplett wiedergegeben zu werden. Sie können aber auch von **www.reglos.de/kindermann/ffx.html** heruntergeladen werden.

```
function nett=newffit(IT,PR,S,T,BTF,BLF,PF)
% NEWFFIT - iterative roots extension of NEWFF

if ~ exist('S')
   disp (['Info: NEWFFIT Created an Output Layer of Size '
num2str(size(PR,1))])
   S=size(PR,1);
end

if S(size(S,2)) ~= size(PR,1)
   disp (['Info: NEWFFIT added an Output Layer to match Input/
Output Size of ' num2str(size(PR,1))])
   S=[S size(PR,1)];
end

if ~ exist('T')
   T=cell(1,size(S,2));
   for i=1:size(S,2)-1
      T(1,i)={'tansig'};
   end
   T(1,size(S,2))={'purelin'};
end

if ~ exist('BTF')
    BTF='trainbfg';
end

if ~ exist('BLF')
    BLF='learngdm';
end

if ~ exist('PF')
    PF='mseit';
end

Si=S;
Ti=T;
O=[zeros(1,size(S,2)-1) ,1];
Oi=O;
for i=1:IT-1
   Si=[Si S];
   Ti=[Ti T];
   Oi=[Oi O];
end
```

```
net=newff(PR,Si,Ti,BTF,BLF,PF);

net.outputConnect=Oi;

net.userdata.note='Iterative Root network';
net.userdata.IT  = IT;
net.userdata.DIM = size(PR,1);
net.userdata.PR  = PR;
net.userdata.S   = S;
net.userdata.T   = T;

if strcmp(PF,'mseit')
 %to show mseit how many iLayers there are...
 net.performParam.IT = IT;
end

nett=net;

function perf = mseit(e,x,pp)
% MSEIT Mean square error with iLayer variance performance funct

% FUNCTION INFO
if isstr(e)
  switch e
    case 'deriv',
  perf = 'dmseit';
    case 'name',
  perf = 'Mean squared error with iLayer regularization';
    case 'pnames',
  perf = {'ratio'};
    case 'pdefaults',
       performParam.ratio = 0.9;
       performParam.IT = 'Must be set to net.userdata.IT!';
    perf = performParam;
    otherwise,
  error('Unrecognized code.')
  end
  return
end

% CALCULATION
if nargin < 3
  if (nargin == 2) & (isa(x,'network') | isa(x,'struct'))
    pp = x.performParam;
 x=getxit(x);
  else
    error('Not enough input arguments');
  end
else
   if min(size(x))==1
      %warning('x not a matrix')
```

```
      x=reshape (x',length(x)/pp.IT,pp.IT);
   end
end

if isa(e,'double'), e = {e}; end

e = cell2mat(e);
perf = sum(sum(e.^2))/prod(size(e));

if pp.IT>1
 xm=nncopy(mean(x')',1,pp.IT);
 perfx=sum(sum((x-xm).^2))/prod(size(x));
 perf = pp.ratio*perf + (1-pp.ratio)*perfx;
end

function d = dmseit(code,e,x,perf,pp)
% DMSEIT - iterative roots extension of DMSE : Mean square error
% with iLayer regularisation performance derivative function.

if nargin < 5, error('Not enough input arguments.'),end

doubleForm = 0;
if isa(e,'double'), e = {e}; doubleForm = 1; end

switch code
  case 'e',
    numElements = prod(size(cell2mat(e)));
    [rows,cols] = size(e);
    d = cell(rows,cols);
m = pp.ratio * 2/numElements;
for i=1:rows
  for j=1:cols
    d{i,j} = e{i,j} * m;
  end
end
if doubleForm, d = cell2mat(d); end

  case 'x',
     if min(size(x))==1
       warning('x not a matrix')
       x=reshape (x',length(x)/pp.IT,pp.IT);
    end
    m = (1-pp.ratio)*2/prod(size(x));
    %Ableitung des Fehlers nach einem Gewicht ist
    %Abweichung des Gewichtes vom Mittelwert aller
    %gekoppelten Gewichte in den iLayern durch
    %die Anzahl der Gewichte.
    d=reshape(nncopy(mean(x')',1,size(x,2))-x,prod(size(x)),1)*m;

  otherwise,
    error(['Unrecognized code.'])
end
```

```
function net = meanit(net)
% MEANIT - iterative root network smoothing routine
if net.userdata.IT>1
 net=setxit(net,mean(getxit(net)')');
end

function W = getxit(net,layers)
% GETXIT - iterativ root extension off GETX
if ~exist('layers'); layers=1:net.userdata.IT; end
l=0;
W=[];
for i=layers %iLayers
   w=[];
   for j=1:size(net.userdata.S,2)
      l=l+1;
      if l==1
         v=net.IW{1,1};
      else
         v=net.LW{l,l-1};
      end
      w=[w;v(:)];
      v=net.b{l,1};
      w=[w;v(:)];
   end
   W=[W w];
end

function net = setxit(net,W)
% SETXIT - iterative roots replacement for SETX
if size(W,2)==1
 W=nncopy(W,1,net.userdata.IT);
end
l=0;
for i=1:net.userdata.IT %iLayers
   p=1;
   for j=1:size(net.userdata.S,2)
      l=l+1;
      if l==1
         s=size(net.IW{1,1}(:),1);
         net.IW{1,1}(:)=W(p:p+s-1,i);
         p=p+s;
      else
         s=size(net.LW{l,l-1}(:),1);
         net.LW{l,l-1}(:)=W(p:p+s-1,i);
         p=p+s;
      end
      s=size(net.b{l,1}(:),1);
      net.b{l,1}(:)=W(p:p+s-1,i);
      p=p+s;
   end
end
```

```
function [new, t] = pretrainit(net,x,y,PreTrain)
% PRETRAINIT - iterative root pretraining routine

if ~exist('y')
  y=x;
  PreTrain=1;
end;

if ~exist('PreTrain')
  PreTrain=2;
end;

if PreTrain ~= 0
 if abs(PreTrain) == 1
   disp('Pre-Training single iLayer to Identity')
   t = x .* sign(gradient(y));
 elseif abs(PreTrain) == 2
   disp('Pre-Training single iLayer to Target')
   t = y;
 elseif abs(PreTrain) == 3
   disp('Pre-Training single iLayer to Intermediate')
   t = y*(1/net.userdata.IT)+x .* sign(gradient(y))* ...
       (1-(1/net.userdata.IT));
 end
 if PreTrain<0
   t = -t;
 end
 net0=newffit(1,net.userdata.PR,net.userdata.S,net.userdata.T,...
              net.trainFcn,'learngdm','mse');
 net0.trainParam.goal   = net.trainParam.goal;
 net0.trainParam.epochs = net.trainParam.epochs;
 net0=train(net0,x,t);

 new=setx(net,nncopy(getx(net0),net.userdata.IT,1));

else

 new=net;

end
```

ANHANG B

Verzeichnis der Bilder und Tafeln

ANHANG C

Literaturverzeichnis

C.1 Iterative Wurzeln und fraktionale Iterationen

Bei der ersten Begegnung mit der Gleichung $f(f(x)) = g(x)$ war es nicht leicht, die entsprechende Literatur zu finden. Fast 200 Jahre alte Arbeiten sind immer noch aktuell, es gibt kein dediziertes Lehrbuch und lediglich eine Handvoll Monografien neben den Zeitschriften. Bei der Suche fanden sich etliche Quellen, die über die hier gegebene Einführung in das Gebiet weit hinausgehen, aber doch für spezielle Anwendungen interessant sein könnten. So kam die z. Zt. wohl vollständigste Literaturliste über iterative Wurzeln zusammen. Da diese trotzdem nur etwa 100 Arbeiten umfasst, sind hier nicht nur die explizit zitierten Stellen angegeben, sondern die komplette Liste als Referenz für eine weitere Beschäftigung mit diesem Gebiet.

Als Empfehlung für den Einstieg sei das Buch „Topics in Iteration Theory“ [Targonski 1981] mit einem Ergänzungsartikel [Targonski 1995] genannt. Das umfassendste Werk ist „Iterative Functional Equations“ [Kuczma 1990], zu dem ebenfalls eine Aktualisierung mit einer Übersicht über neueste Ergebnisse vorliegt [Baron 2001].

Eine aktualisierte Onlineversion mit vielen Links auf herunterladbare Artikel, Homepages der Autoren usw. liegt auf **www.reglos.de/kindermann/ffx.html**.

[1] J. Aczél & J. Dhombres, *Functional equations in several variables.* Encyclopedia of Mathematics and its Applications, Cambridge University Press 1989

[2] R. Aldrovandi & L.P. Freitas, *Continuous Iteration of Dynamical Maps.* J. Math. Phys. 39 (1998), 5324-5336

[3] C. Babbage, *Essay towards the Calculus of functions.* Phil. trans. Royal Soc. London 105 (1815), 389-424

[4] C. Babbage, *Examples of the solution of functional equations.* Cambridge 1820

[5] M. Bajraktarevic, *Solution générale de l'équation fonctionelle* $f^N(x) = g(x)$. Publ. Inst. Math. Beograd (N.S.) 5(19) (1965), 115-124

[6] I.N. Baker, *The iteration of entire transcendental functions and the solution of the functional equation* $f(f(z) = F(z)$. Math. Ann. 129 (1955), 174-180

[7] I.N. Baker, *Zusammensetzung ganzer Funktionen.* Math. Zeitschr. 69 (1958), 121-163

[8] K. Baron & W. Jarczyk, *Recent results on functional equations in a single variable, perspectives and open problems.* Aequationes Math. 61. (2001), 1-48

[9] L. Bartlomiejczyk, *Iterative roots with big graph.* Abstract of Summer symposium in Real Analysis, XXIV, Real Anal. Exchange

[10] J. Beránek, *On Tabors problem concerning a certain quasi - ordering of iterative roots of functions*. Aequationes Math. 39. (1990), 1-5

[11] A. Blokh, E. Coven, M. Misiurewicz & Z. Nitecki, *Roots of continuous piecewise monotone maps of an interval.* Acta Math. Univ. Comenianae, (1991), 3-10

[12] A.M. Blokh, *The set of all iterates is nowhere dense in C ([0,1],[0,1]).* Trans. Amer. Math. Soc. 322, (1992), 787-798

[13] J.R. Blum & N. Friedmann, *On commuting transformations and roots.* Univ. New Mexico, Techn. Rep. No. 98, 1965

[14] U.T. Bödewadt, *Zur Iteration reeller Funktionen.* Math. Zeitschr. 49 (1943), 497-516

[15] S. Bogatyi, *On the nonexistence of iterative roots*. Topology and its applications. 76 (1997), 97-123

[16] S. Bogatyi, *A topological view on a problem of formal series.* Aequationes Math. 56 (1998), 18-22

[17] K. Briggs, *Formal power series solution of functional equations.* MapleTech 2, no. 2 (1995), 48-51

[18] K.M. Brigs, T.W. Dixon & G. Szekeres, *Analytic solutions of the Cvitanovic-Feigenbaum and Feigenbaum-Kadanoff-Shenker equations.* Int'l. J. of Bifurcation and Chaos 8 (1998), 347-357

[19] E.N. Bronshtein, On an iterative square root of a quadratic trinomial (Russian). In: Geometric Problems in the theory of functions and sets. 24-27, Kalinin Gos. Univ., Kalinin, 1989

[20] K. Brown, *f(f(x)) = exp(x) generates the Multinomials.* Draft on website http://www.seanet.com/~ksbrown/kmath507.htm

[21] R. Brown, *On solving nonlinear functional, finite difference, composition and iterated equations.* Fractals 7 (1999), 277-282

[22] E. Castillo & A. Iglesias, *A package for symbolic solution of real functional equations of real variables.* Aequationes Math. 54 (1997), 181-198

[23] B. Choczewski, *Problem 27.* Report of meeting. The Twenty-eight International Symposium on Functional Equations (Graz-Mariatrost, 1990), Aequationes Math. 41 (1991), 296

[24] B. Choczewski & M. Kuczma, *On iterative roots of polynomials.* European conference of iteration theory - Lisboa '91 (1992), 59-67

[25] J.P.R. Christensen, *Topology and Borel Structure.* North-Holland Math. Studies 10, Amsterdam, 1974

[26] L. Comtet, *Advanced Combinatorics.* Reidel Publishing, Dordrecht, 1974

[27] Wun-Seng Chou, *Iterative roots of constant polynomials over finite fields*. Meeting of the Amer. Math. Soc., Las Vegas, 1999

[28] P.J. Davis, *Interpolation and Approximation.* Blaisdell Publishing Company, New York, 1963

[29] G. Derfel & R. Schilling, *Spatially chaotic configurations and functional equations with rescaling.* J. Phys. A: Math. Gen. 29 (1996), 4537-4547

[30] K.D. Dikof & R. Graw, *A strictly increasing real function with no iterative roots of any order.* Remark at the 1980 International Symposium on Functional Equations.

[31] K.B. Dunn & R. Lidl, *Iterative roots of functions over finite fields.* Math. Nachr. 115 (1984), 319-329

[32] P. Erdös & E. Jabotinsky, *On Analytic Iteration.* J. Analyse Math. 8 (1960/61), 361-376

[33] G.M. Ewing & W.R. Utz, *Continuous solutions of $f^n(x) = f(x)$.* Can. J. Math. 5 (1953), 101-103

[34] Faá di Bruno, *Sullo sviluppo delle funzioni.* Annali di Scienze Matematiche et Fisiche di Tortoloni 6 (1855), 479-480

[35] M.J. Feigenbaum, *Quantitative Universality for a Class of Nonlinear Transformations.* J. of Statistic Physics 19 (1978), 25-52

[36] B. Gawel, *On the uniqueness of continuous solutions of functional equations.* Ann. Polon. Math. LX.3 (1995), 231-239

[37] V. Goryainov, *Koenigs function and fractional iteration of probability generating functions.* XVIII Nevanlinna Colloquium, University of Helsinki, 2000 www.math.helsinki.fi/~analysis/NevanlinnaColloquium/abstracts/posters/gory.ps

[38] P.I. Haidukov, *On searching a function from a given iterate.* Uc. Zap. Buriatsk. Ped. Inst. 15 (1958), 361-376

[39] P.R. Halmos, *Square roots of masure preserving transformations.* Am. J. Math. 64 (1942), 153-166

[40] H.J. Hamilton, *Roots of equations by functional iteration.* Duke. Math. J. 13 (1946), 113-121

[41] G.H. Hardy, E. Cunningham, *Orders of infinity.* Cambridge Tracts in Mathematics 12 (1924), 31

[42] C. Horowitz, *Iterated logarithms of entire functions*. Israel J. Math. 29 (1978), 31-42

[43] P.D. Humke & M. Laczkovich, *The Borel structure of iterates of continuous functions.* Proc. Edinburgh Math. Soc. (2) 32 (1989), 483-494

[44] K. Iga, *Continuous half-iterates of functions*, Manuscript. URLs http://math.stanford.edu/~iga or http://math.pepperdine.edu/~kiga

[45] R. Isaacs, *Iterates of fractional order.* Canad. J. Math. 2 (1950), 409-416

[46] R. Isaacs, *On Fractional Iteration*. Technical Report No. 320, Department of Mathematical Sciences, The John Hopkins University, November 1979

[47] E. Jabotinsky, *Analytic iteration.* Trans. Amer. Math. Soc. 108 (1963), 457-477

[48] W. Jarczyk, *A recurrent method of solving iterative functional equations.* Prace Nauk. Uniw. Slask. Katowic. 1206, 1991

[49] W. Jarczyk, *Babbage equation on the circle.* Report of Meeting, The Thirty-eighth International Symposium on Functional Equations, 2000, Noszvaj, Hungary. Aequationes Math. 61 (2001), 281-320

[50] L. Kindermann, *Computing Iterative Roots with Neural Networks.* Proc. Fifth Int'l Conf. on Neural Information Processing - ICONIP'98, Vol. 2, Kitakyushu (1998), 713-715

[51] L. Kindermann & A. Lewandowski, *A Comparison of Different Neural Methods for Solving Iterative Roots.* Proc. Seventh Int'l Conf. on Neural Information Processing - ICONIP'2000, Taejon (2000), 565-569

[52] H. Kneser, *Reelle analytische Lösungen der Gleichung* $\varphi(\varphi(x)) = e^x$ *und verwandter Funktionalgleichungen.* J. reine angew. Math. 187 (1950), 56-67

[53] J. Kobza, *Iterative functional equation x(x(t))=f(t) with f(t) piecewise linear.* Journal of Computational and Applied Mathematics 115 (2000), 331-347

[54] A.R. Kräuter, *Wurzeln und analytische Iteration formal-biholomorpher Abbildungen.* Berichte Math. Stat. Forschungszentrum Graz 123, 1980

[55] U. Krengel und H. Michel, *Über Wurzeln ergodischer Transformationen.* Math. Z. 96 (1967), 50-57

[56] D. Kroll, *Grundlagen Nichtlinearer Analyse: Differential und Differenzengleichungen.* In: Nichtlineare Dynamik in kondensierter Materie, Kernforschungsanlage Jülich, 1983

[57] M. Kuczma, *On the functional equation* $\varphi^n(x) = g(x)$. Ann. Polon. Math. 11 (1961), 161-175

[58] M. Kuczma, *On the Schröder equation.* Rozprawy Matematyczne, Instytut Matematyczny Polskiej Akademii Nauk, Warszawa, 1963

[59] M. Kuczma & A. Smajdor, *Fractional Iteration in the class of convex functions.* Bull. Acad. Polon. Sci., Ser. sci. math. astronom. phys. 16 (1968), 717-720

[60] M. Kuczma, *Functional Equations in a Single Variable,* Monografie Matematyczne, Polska Akademia Nauk, PWN-Polish Scientific Publishers, Warsaw, 1968

[61] M. Kuczma, *Fractional iteration of differentiable functions.* Ann. Polon. Math. 22 (1969), 217-227

[62] M. Kuczma, *Fractional iteration of differentiable functions with multiplier zero.* Commentationes Math. 14 (1970), 35-39

[63] M. Kuczma & A. Smajdor, *Regular fractional iteration.* Bull. Acad. Polon. Sci., Ser. sci. math. astronom. phys. 19 (1971), 203-207

[64] M. Kuczma, *Regular fractional iteration of convex functions.* Ann. Polon. Math. 38 (1980), 95-100

[65] M. Kuczma, B. Choczewski & R. Ger, *Iterative functional equations*. Encyclopedia of Mathematics and its Applications, Vol. 32, Cambridge University Press, Cambridge, 1990

[66] M. Kuczma, *On a functional equation occurring in astrophysics.* Mathematica Pannonica 3/2 (1992), 17-27

[67] R. Leipnik & R. Oberg, *Subvex functions and Bohr's uniqueness theorem.* Amer. Math. Monthly 74 (1967), 1093-1094

[68] Z. Lesniak, *Constructions of fractional iterates of sperner homeomorphisms of the plane.* In: W. Förg-Rob, Iteration Theory, Singapore 1996, 182-192

[69] He Lianfa & New Dongxiao, *The iterative roots for a class of selfmapping on S^1.* Jour. Math. Res. & Exp. 11 (1991), 305-310

[70] R. Liedel, On Phantom Roots and Imbeddings. In: W. Förg-Rob, Iteration Theory, Singapore 1996, 193-198

[71] J.C. Lillo, *The functional equation $f^n(x) = g(x)$*. Arkiv för Mat. 5 (1965), 357-361

[72] J.C. Lillo, *The functional equation $f^2(x) = g(x)$.* Ann. Polon. Math. 19 (1967), 123-135

[73] L.S.O. Liverpool, *Fractional iteration near a fix point of multiplier 1.* J. London Math. Soc. 41 (1979)

[74] S. Lojasiewicz, *Solution générale de l'équation fonctionelle f(f(...f(x)...))) = g(x).* Annales de la Societé Plonaise de Mathematique 24 (1951), 88-91

[75] J.L. Massera & A. Petracca, *On the functional equation f(f(x)) = 1/x.* Revista de la Union Mat. Argentina 11 (1946), 206-211

[76] P.J. McCarthy, *Ultrafunctions, projective function geometry, and polynomial functional equations.* Proc. London Math. Soc. (3) 53 (1986), 321-339

[77] A. McNaughton, *Generalised iterative roots*, Abstracts of the New Zealand Mathematics Colloquium 2000, ACMA Abstracts, 2000

[78] P.B. Miltersen, N.V Vinodchandran & O. Watanabe, *Super-polynomial versus half-exponential circuit size in the exponential hierarchy.* Research Report c-130, 1999. Dept. of Math and Comput. Sc., Tokyo Inst. of Tech. and BRICS Report Series RS-99-4, Dept. of Computer Science, Univ. Aarhus, 1999

[79] C. Mira & S. Müllenbach, *Sur l'iteration fractionnaire d'un endomorphisme quadratique.* Comptes rendus Acad. Sci. Paris (1) Math. 297, (1983), 369-372

[80] T. Narayaninsamy, *On the fractional iteration of endomorphisms.* In: European Conference on Iteration Theory, eds. J. P. Lampreia, J. Llibre, C. Mira, J. Sousa Ramos, G. Targonski, Singapore, 1991

[81] T. Narayaninsamy, *On a class of fractional iterates.* Int. J. of Bifurcation and Chaos 6 (1996), 55 - 67

[82] T. Narayaninsamy, *Fractional iterates for n-dimensional maps*. J. Applied Mathematics and Computation 98 (1999), 261-278

[83] T. Narayaninsamy, *A connection between fractional iteration and graph theory.* J. App. Math. Comp. 107 (2000), 181-202

[84] D.S. Ornstein, *Ergodic theory, randomness and dynamical systems.* Yale University Press, 1974

[85] T. Powierza, *Set-valued iterative square roots of bijections.* Bull. Pol. Ac. Math. 47 (1999)*, 377-383*

[86] L. Reich, *Analytische und fraktionelle Iteration formal-biholomorpher Abbildungen.* Jahrbuch Überblicke Mathematik 1979, Bibliographisches Institut, 1979

[87] L. Reich, *On power series transformations in one indeterminate having iterative roots of given order and with given multiplier.* European Conf. Iteration Theory - ECIT, Lisbon (1991), 210-216

[88] L. Reich, *On the Distribution of Formal Power Series Transformations with Respect to Embeddability in the Order Topology.* Sitzungsberichte der Österreichischen Akademie der WIssenschaften, Abt. II 208 (1999), 97-114

[89] B.A. Reznick, *A uniqueness criterion for fractional iteration.* Ann. Polon. Math. 30 (1974), 219-224

[90] R.E. Rice, *Fractional iterates.* PhD Thesis, University of Massachusetts, Amherest, 1977

[91] R.E. Rice, *Iterative square roots of Cebysev polynomials.* Stochastica 3 (1979), 1-14

[92] R.E. Rice, B. Schweizer & A. Sklar, *When is f(f(z)) = az^2+bz+c for all complex z?* Amer. Math. Monthly 87 (1980), 252-263

[93] G. Riggert, *n-te iterative Wurzeln von beliebigen Abbildungen.* Aequationes. Math. 14 (1976), 208

[94] G. Riggert, *Note on n-th iterative roots of mappings of a finite set into itself.* Aequationes Math. 15 (1977), 288

[95] R. Resch, F. Stenger & J. Waldvogel, *Functional equations related to the iteration of functions.* Aequationes Math. 60 (2000), 25-37

[96] R. Schilling, *Spatially chaotic structures*. In: H. Thomas, Nonlinear Dynamics in Solids, Springer-Verlag, Berlin, 1992, 213-241

[97] E. Schröder, *Über iterierte Funktionen.* Math. Ann. 3 (1871), 296-322

[98] M.R. Schröder, *Fractals, Chaos, Power Laws*. Freeman, New York 1991

[99] J. Schwaiger, *Roots of formal power series in one variable*. Aequationes Math. 29 (1985), 40-43

[100] B. Schweizer & A. Sklar, *Invariants and equivalence of polynomials under linear conjugacy.* Contributions to General Algebra 6 (1988), 253-257

[101] K. Simon, *Some dual statements concerning Wiener measure and Baire category.* Proc. Amer. Math. Soc. 106 (1989), 455-463

[102] K. Simon, *Typical continuous functions are not iterates.* Acta Math. Hung. 55 (1990), 133-134

[103] A. Sklar, *Canonical decompositions, stable functions and fractional iterates.* Aequat. Math. 3 (1969), 118-129

[104] A. Smith, *Is there any way to approximate the solution of f(x) given f(f(x)) = g(x)?* Argonne National Laboratory, Division of Educational Programs. Newton BBS: Ask A Scientist Mathematics Archive (1993) www.newton.dep.anl.gov/newton/askasci/1993/math/MATH023.HTM

[105] L. Smolarek, *The formula of fractional iteration of function differentiable at its fixed point.* Zeszyty Nauk. Mat. Uniw. Gdans. 7 (1987), 99-107

[106] J. Spanier & K.B. Oldham, *The fractional calculus.* New York: Academic Press, 1974

[107] G. Szekeres, *Regular iterations of real and complex functions.* Acta Math. 100 (1958), 203-258

[108] G. Szekeres, *Fractional iteration of exponentially growing functions.* J. Austral. Math. Soc. 2 (1961/62), 301-320

[109] G. Targonski, *An Iteration theoretical approach to the concept of time.* Colloques Internationaux du C.N.R.S. 229, Transformations ponctuelles et leurs applications, Toulouse (1973), 259-271

[110] G. Targonski, *Topics in iteration theory.* Vandenhoeck und Ruprecht, Göttingen, 1981

[111] G. Targonski, *Phantom Iterates and Liedl's pilgerschritt transformation*. In: W. Förg-Rob, Iteration Theory, Singapore 1996, 284

[112] G. Targonski, *Progress of iteration theory since 1981.* Aequationes Math. 50 (1995), 50-72

[113] H. Töpfer, *Komplexe Iterationsindizes ganzer und rationaler Funktionen*. Math. Ann. 121 (1949), 191-222

[114] P.L. Walker, *The exponential of iteration of e^x-1.* Proc. Amer. Math. Soc. 110 (1990), 611-620

[115] P. Walker, *Infinitely differentiable generalized logarithmic and exponential functions*. Mathematics of Computation 57 (1991), 723-733

[116] G.J. Whitrow, *The Natural Philosophy of Time.* Thomas Nelson and Sons Ltd. London and Edinburgh, 1961

[117] M.C. Zdun, *Continuous iteration semigroups.* Boll. Un. Mat. Ital. 14 A (1977a), 65-70

[118] M.C. Zdun, *Differentiable fractional iteration.* Bull. Acad. Sci. Polon. Sér. Sci. Math. Astronom. Phys. 25 (1977b), 643-646

[119] M.C. Zdun, *On differentiable iteration groups.* Publ. Math. Debrecen 26 (1979), 105-114

[120] M.C. Zdun, *Regular fractional iterations.* Aequationes Math. 28 (1985), 73-79

[121] M.C. Zdun, *On iterative roots of homeomorphisms of the circle.* Bull. Pol. Ac. Sci. Math. 48/2 (2000), 203-214

[122] W. Zhang, *A generic property of globally smooth iterative roots.* Scientia Sinica A, 38 (1995), 267-272

[123] G. Zimmermann, *Über die Existenz iterativer Wurzeln von Abbildungen.* Dissertation, Marburg/Lahn 1978

C.2 Neuronale Netze

[124] J. Anlauf, A. Gröber, B. Knapp, *Neurocomputer SYNAPSE1 für vernetzte Lösungen mit künstlichen Neuronalen Netzen*. e&i 7/8, 1995

[125] C.M. Bishop, *Neural Networks for Pattern Recognition.* Oxford University Press1995

[126] R. Böhm, *ECANSE - How human reasoning can be imitated.* Siemens-Review 6, 1995

[127] A.M. Chen, H. Lu & R. Hecht-Nielsen, *On the Geometry of feedforward neural network error surfaces.* Neural Computation 5 (1993), 910-927

[128] Y. LeCun, L. Bottou, G.B. Orr & K.R. Müller, *Efficient Backprop.* In: G.B. Orr and K.R. Müller, Neural Networks:Tricks of the Trade, Lecture Notes in Computer Science Vol. 1524, Springer, 1998, 9-50

[129] D.T. Davies, Z. Chen, J.N. Hwang, L. Tsang, & A.T.C. Chang, *Retrival of snow parameters by iterative inversion of a neural network.* IEEE Trans. on Geoscience and Remote Sensing 31 (1993), 842-852

[130] G.W. Davis and M.L. Gasperi, *ANN Modelling of Volterra Systems.* Proc. Int'l Joint Conf. on Neural Networks - IJCNN'91, Seattle (1991), II 727-734

[131] J.L. Elman, *Finding structure in time.* Cognitive Science 14 (1990), 179-211

[132] M. Gälli, *Integration von neuronalen Netzen in mathematische Prozeßmodelle mit objektorientierten Methoden.* Diplomarbeit, Lehrstuhl für Künstliche Intelligenz, Erlangen, 1996

[133] S. He, K. Reif & R. Unbehauen, *Control of continuous-time nonlinear systems using neural networks.* Proceedings of the International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing - NICROSP, 1996

[134] K. Hornik, M. Stinchcombe & H. White, *Multi-layer Feedforward Networks are Universal Approximators.* Dans Neural Networks. Vol. 2, pp. 359-366, 1989

[135] J. Kindermann & A. Linden, *Inversion of neural networks by gradient descent.* Parallel Computing, 14(3) (1990), 277-286

[136] L. Kindermann, *An Addition to Backpropagation for Computing Functional Roots.* Proc. Int'l ICSC/IFAC Symp. on Neural Computation - NC'98, Vienna (1998), 424-427

[137] L. Kindermann & T.P. Trappenberg, *Modeling time-varying processes by unfolding the time domain.* Proc. Int'l Joint Conf. on Neural Networks (IJCNN'99), Washington DC, (1999)

[138] L. Kindermann, A. Lewandowski, M. Tagscherer & P. Protzel, *Computing Confidence Measures and Marking Unreliable Predictions by Estimating Input Data Densities with MLPs*. Proc. Sixth Int'l Conf. on Neural Information Processing (ICONIP'99), Perth (1999), 91-94

[139] S. Kröner & R. Moratz, *Capacity of multilayer networks with shared weights.* Proc. Int'l Conf. on Artificial Neural Networks - ICANN'96 (1996), 543-550

[140] B. Lautrup, L. Hansen, I. Law, N. Morch, C. Svarer, S. Strother, *Massive Weight Sharing: A cure for extremely ill-posed problems.* Workshop in Brain Research: From Tomography to Neural Networks, Julich (1994), 137-148

[141] A. Linden & J. Kindermann, *Inversion of multilayer nets.* Proc. Int'l Joint Conf. on Neural Networks - IJCNN, Washington (1989), 425-430

[142] A. Linden, *SESAME - ein objekt und datenflussorientierter Simulator für Modelle der Neuroinformatik und angrenzender Gebiete.* Dissertationen zur künstlichen Intelligenz, Bd 59, Sankt Augustin, 1995

[143] S. Nowlan & G. Hinton, *Simplifying neural networks by soft weight sharing.* Neural Computation 4 (1992), 473-493

[144] A. Pinkus, *Approximation theory of the MLP model in neural networks.* Acta Numerica (1999), 143-195

[145] D.E. Rumelhart, G.E. Hinton & R.J. Williams, *Learning Internal Representations by Error Propagation.* In: D.E. Rumelhart, J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1,* London, The MIT Press, 1986, 318-362

[146] J. Shawe-Taylor, *Introducing Invariance: a principled approach to weight sharing.* Proc. IEEE Intl. Conf. on Neural Networks, Orlando (1994), 345-349

[147] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano & K.J. Lang, *Phoneme recognition using time-delay neural networks.* IEEE Trans. Acoustics, Speech, Signal Processing 37 (1989), 328-339

[148] P.J. Werbos, *Backpropagation through time: What it is and how to do it.* Proc. of the IEEE, 78 (1990), 1550-1560

[149] D. Zhang & Y. Hen Hu, *Global and local weight sharing.* Proc. Int. Conf. on Neural Networks - ICNN (1996),1494-1498

[150] Thurner E.: Environment for Computer Aided Neural Software Engineering, e&i 7/8, 1995

C.3 Walzen von Stahl

[151] G. Beisemann, *Theoretische Untersuchung der mechanisch einstellbaren Bereiche für die Walzspaltform an unterschiedlichen Walzwerksbauarten.* Dissertation TH Aachen, Umformtechnische Schriften, Band 7, Verlag Stahleisen mbH, Düsseldorf, 1987

[152] R. Isermann, *Identifikation dynamischer Systeme 1*. Springer Berlin, Heidelberg, New York 1992

[153] L. Kindermann, P. Protzel, F. Schmid und O. Gramckow, *Verfahren und Einrichtung zur Bestimmung eines Zwischenprofils eines Metallbandes.* Patentschrift DE 1998/0248D, 1998

[154] L. Kindermann, P. Protzel, F. Schmid & O. Gramckow, *Process and device for determining an intermediate section of a metal strip*. Internationale Patentschrift WO 99/42232, 1999

[155] J. Larkiola, *Experience on Using Neural Network on Modelling/Optimization of Strip Production.* BAMFAC Metal Forming and Cutting Seminar, VTT, Espoo, 1996

[156] D. Linfhoff, G. Sörgel, O. Gramckow, und K.D. Klode, *Erfahrungen beim Einsatz neuronaler Netze in der Walzwerksautomatisierung.* Stahl und Eisen 114 (1994), 49-53

[157] T. Martinetz , P. Protzel, O. Gramckow & G. Sörgel, *Neural network control for steel rolling mills.* In B. Kappen and S. Giele, eds, Neural Networks: Artificial intelligence and industrial application, Springer, Berlin, 1995

[158] T. Martinetz, O. Gramckow, P. Protzel, und G. Sörgel, *Neuronale Netze zur Steuerung von Walzstraßen.* atp - Automatisierungstechnische Praxis 38, (1996) 28-42

[159] C. Menéndez, J.B. Ordieres & F. Ortega, *The importance of information pre-processing in the improvement of neural network results.* International Journal on Expert Systems and Neural Networks, 13-2 (1996), 95-103

[160] D. Neumerkel, *Neuronale Regelung von Walzgerüsten.* NewsServer 1 (1996), 23-24

[161] J.B. Ordieres, C. Menéndez, F. Ortega, *Width control in semicontinuous rolling mills: A neural approach*. Proc. of the ICNN (1996)

[162] T. Poppe, D. Obradovic & Martin Schlang, *Neural networks: Reducing energy and raw materials requirements.* Siemens Review, 1995

[163] P. Protzel, L. Kindermann, M. Tagscherer & A. Lewandowski, *Adaptive Systemidentifikation mit Neuronalen Netzen zur Profilsteuerung in Walzwerken.* VDI Berichte 1381 (1998), 347-359

[164] P. Protzel, L. Kindermann, M. Tagscherer & A. Lewandowski, *Abschätzung der Vertrauenswürdigkeit von Neuronalen Netzprognosen bei der Prozessoptimierung*. VDI Bericht 1626 zur Fachtagung der VDI/VDE Gesellschaft für Mess- und

Automatisierungstechnik: Computational Intelligence, Baden-Baden (2000), 335-339

[165] J. Rieckmann, *Berechnung von Bandprofil und Planheit beim Kaltwalzen auf Sechs-Walzen-Walzwerken.* Dissertation TH Aachen, Umformtechnische Schriften, Band 21, Verlag Stahleisen mbH, Düsseldorf, 1989

[166] M. Röscheisen, R. Hoffmann & V. Tresp, *Neural control for rolling mills: Incorporating domain theories to overcome data defiency.* Advances in Neural Information Processing Systems, Morgan Kaufmann (1992), 659-666

[167] D. Sbarbaro-Hofer, D. Neumerkel & K. Hunt, *Neural control of a steel rolling mill.* IEEE Control Systems Magazine 13 (1993), 69-75

ANHANG D

Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützung von folgenden Personen erhalten:

- Prof. Dr. Peter Protzel stellte das Thema dieser Arbeit.
- Dr. rer. pol. Achim Lewandowski half mit zahlreichen Anmerkungen und Diskussionen zu mathematischen Inhalten.
- Dipl. Inf. Carsten Schneider implementierte das „Weight Sharing“ in FAST.

Weitere Personen waren an der geistigen Herstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich nicht die Hilfe eines Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Erlangen, 1. Juli 2001

Lars Kindermann

ANHANG E Thesen

1. Iterative Wurzeln einer Funktion f sind die Lösungen φ der Funktionalgleichung $\varphi(\varphi(x)) = f(x)$, Fraktionale Iterationen werden definiert durch $\varphi^n = f^m$. Sie stellen ein altes mathematisches Problem dar, das erstmals 1815 auftauchte und einfach zu erklären, aber sehr schwierig zu lösen ist. Sie stellen die natürliche Erweiterung der Operationen „Wurzel“ und „Potenz“ auf die Funktionalalgebra dar. Gleichzeitig sind sie das inverse Problem der Iteration. Die bisherige Literatur zu diesem Thema konzentriert sich auf abstrakt formale Fragestellungen, Existenz, Eindeutigkeit, Regularisierungen, Struktur der Lösungsräume etc. Die konkrete Berechnung von iterativen Wurzeln gegebener Funktionen wird - wenn überhaupt - eher am Rande angesprochen und ist bisher in keiner Weise, z.B. durch Formelsammlungen, Algorithmen oder Software, unterstützt. Es gibt eine Reihe von Problemen u.a. aus Technik, Ökonomie und Biologie, die die Berechnung von iterativen Wurzeln erfordern, deren Lösung jedoch durch die mathematischen Schwierigkeiten behindert wird. Die Notwendigkeit für Anwender, sich in eine vorwiegend abstrakte Literatur einzuarbeiten, ist nicht selten abschreckend.

2. Experimentelle Daten von dynamischen Vorgängen liegen immer in diskreter Form vor, da durch Begrenzung der Abtastraten und des Auflösungsvermögens eine wirklich kontinuierliche Messung nicht möglich ist. Um daraus ein kontinuierliches Modell, z.B. eine Differentialgleichung zu erstellen, müssen a priori Modellgleichungen eingeführt werden, deren Parameter an die Daten angepasst werden. Die reelle Iteration experimentell gewonnener Zusammenhänge finiter Auflösung ist dagegen ein modellfreier Ansatz zur Einbettung dieser diskreten Dynamik in ein Kontinuum, der lediglich eine Translationsinvarianz des Prozesses voraussetzt. Fraktionale Iteration bietet dafür eine gute Näherung.

3. Neuronale Netze sind bei vielen datenbasierten Modellierungsproblemen eine Methode der Wahl. Sie erlauben eine Modellbildung bei nichtlinearen Zusammenhängen, fehlerbehafteten oder unvollständigen Daten. Modellgleichungen sind

nicht erforderlich. Der Nachteil liegt in ihrem Black-Box Charakter. Die Methode, mit mehrschichtigen strukturierten Netzen sowohl einen funktionellen Zusammenhang zu modellieren, und gleichzeitig die Tatsache zu berücksichtigen, dass dieser Zusammenhang auf einer iterierten Abbildung basiert, löst somit zwei Anwendungsprobleme gleichzeitig.

4. Es gibt eine Reihe von technischen Verfahren, die aus der mehrfachen Anwendung des gleichen Prozesses bestehen. Sind nur Daten vorhanden, die den Zusammenhang zwischen Anfangs- und Endzustand beschreiben, erlaubt die iterative Wurzel dieser Funktion auch die Modellierung des Einzelschrittes.

5. Ein modellfreier Ansatz zur Zwischenprofilbestimmung in Stahlwalzwerken ist nicht möglich. Die Iterationsgleichung hat auf den gegebenen Datensätzen keine eindeutige Lösung. Es werden verschiedene Gerüstmodelle gefunden, die zum gleichen Endprofil, aber zu unterschiedlichen Zwischenprofilverläufen führen. Zur eindeutigen Bestimmung der Zwischenprofile ist ein mathematisches Modell des Walzprozesses zwingend notwendig. In Ermangelung eines korrekten Modells ist die lineare Approximation die “einfachste” Annahme. Sie liefert Endprofile mit einer dem bisherigen Modell vergleichbaren Genauigkeit, bei Umstellungsbändern ist sie sogar exakter. Die berechneten Zwischenprofilverläufe weisen einen etwas anderen Verlauf auf als bisher.

6. Eine neuronale Systemmodellierung (direkte Abbildung ohne Iteration) für die gesamte Walzstraße mit einem neuronalen Online-Verfahren liefert bessere Endprofilprognosen. Die Kombination beider Verfahren liefert somit ein Modell, das hohe Endprofilvorhersagegenauigkeit mit eindeutigen Zwischenprofilverläufen liefert.

7. Die Methode zur Berechnung von fraktionalen Iterationen bietet einen Ansatz, auch andere Funktionalgleichungen auf neuronale Netze abzubilden und dadurch deren Lösungen zu bestimmen.

ANHANG F

Lebenslauf

Lars Kindermann

- 1963 Geboren in Uelzen, Niedersachsen
- 1969 Einschulung in Holdenstedt, Kreis Uelzen
- 1982 Abitur in Northeim, Niedersachsen
  Studium der Physik in Göttingen
- 1987 Stipendiat am Max Planck Institut für biophysikalische Chemie in Göttingen, Abteilung für Neurobiologie bei Prof. Otto D. Creutzfeldt
- 1990 Cephalos Gesellschaft für Automatisierungstechnik mbH, Papenburg
- 1992 Diplom in Physik bei Prof. Manfred R. Schroeder, Göttingen. Thema: Der Zusammenhang von Struktur und Funktion kleiner Neuronaler Netzwerke in auditorischen Cortex der Katze
- 1995 Wissenschaftlicher Mitarbeiter am Bayerischen Forschungszentrum für wissensbasierte Systeme, Universität Erlangen in der Forschungsgruppe Neuronale Netze und Fuzzy Logik
- 2001 Promotion an der TU Chemnitz bei Prof. Peter Protzel
  Mitarbeiter am RIKEN Brain Science Institute in Tokyo im “Lab for Mathematical Neuroscience” bei Prof. Shun-Ichi Amari

Aktualisierte Informationen sind auf www.reglos.de/kindermann zu finden.

F.1 Liste der Veröffentlichungen

Viele der Artikel sind im Internet unter www.reglos.de/kindermann direkt verfügbar.

[1] L. Kindermann, H. Redies, U. Sieben, *Real-Time Analysis of the Activity of Small Neuronal Networks Recorded with One Multi-Unit Electrode.* In: New Frontiers in Brain Research, Thieme, Göttingen 1987

[2] L. Kindermann, *Der Zusammenhang von Struktur und Funktion kleiner neuronaler Netzwerke im auditorischen Cortex der Katze.* Diplomarbeit, Max Planck Institut für biophysikalische Chemie, Göttingen 1991

[3] L. Kindermann, H. Redies, S. Brandner, O.D. Creutzfeld, *The Relation between Synaptic Structure and Functional Properties of Small Neuronal Networks in Cat Auditory Cortex.* In: Gene - Brain - Behaviour, Thieme, Göttingen 1993

[4] U. Ziemann, S. Kastner, L. Kindermann, O.D. Creutzfeldt, *Two-Tone Inhibition in Neurons of the Cat's Medial Geniculate Body.* Eur J Neurosci Suppl. 5, (1993), 145-146

[5] P. Protzel, L. Kindermann, M. Tagscherer, A. Lewandowski, *Adaptive Systemidentifikation mit Neuronalen Netzen zur Profilsteuerung in Walzwerken.* In: Computational Intelligence: Neuronale Netze, Evolutionäre Algorithmen, Fuzzy Control im industriellen Einsatz, VDI Berichte 1381, VDI Verlag, Düsseldorf 1998, 347-359

[6] L. Kindermann, *An Addition to Backpropagation for Computing Functional Roots.* Proceedings of the International ICSC/IFAC Symposium on Neural Computation (NC'98), Vienna 1998, 424-427

[7] L. Kindermann, *Computing Iterative Roots with Neural Networks.* Proc. Fifth Int'l Conf. on Neural Information Processing - ICONIP'98, Vol. 2, Kitakyushu (1998), 713-715

[8] L. Kindermann, T.P. Trappenberg, *Modeling time-varying processes by unfolding the time domain.* Proceedings of the International Joint Conference on Neural Networks (IJCNN'99), Washington DC, 1999

[9] L. Kindermann, A. Lewandowski, M. Tagscherer, P. Protzel, *Computing Confidence Measures and Marking Unreliable Predictions by Estimating Input Data Densities with MLPs.* Proceedings of the Sixth International Conference on Neural Information Processing (ICONIP'99), Perth 1999, 91-94

[10] M. Tagscherer, L. Kindermann, A. Lewandowski, P. Protzel, *Overcome neural limitations for real world applications by providing confidence values for network predictions.* Proceedings of the Sixth International Conference on Neural Information Processing (ICONIP'99), Perth 1999, 520-525

[11] A. Lewandowski, M. Tagscherer, L. Kindermann, P. Protzel, *Improving the Fit of Locally Weighted Regression Models.* Proceedings of the Sixth International Conference on Neural Information Processing (ICONIP'99), Perth 1999, 371-374

[12] Fachausschuss 5.21 der VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik GMA, *Künstliche Neuronale Netze in der Automatisierungstechnik - Begriffe und Definitionen.* VDI-Richtlinie: VDI/VDE 3550 Blatt 1, Beuth Verlag, Berlin1999

[13] P. Protzel, L. Kindermann, M. Tagscherer, A. Lewandowski, *Abschätzung der Vertrauenswürdigkeit von Neuronalen Netzprognosen bei der Prozessoptimierung.* In: VDI Bericht 1626 zur Fachtagung der VDI/VDE Gesellschaft für Mess- und Automatisierungstechnik: Computational Intelligence, Baden-Baden 2000, 335-339

[14] P. Protzel, A. Lewandowski, L. Kindermann, M. Tagscherer, B. Herrnberger, *AENEAS: Anwendung und Entwicklung Neuronaler Verfahren zur Autonomen Prozess-Steuerung.* Abschlussbericht BMBF Verbundprojekt Förderkennzeichen: 01 IN 505 B, 2000

[15] L. Kindermann, A. Lewandowski, P. Protzel, *A Comparison of Different Neural Methods for Solving Iterative Roots.* Proc. Seventh Int'l Conf. on Neural Information Processing - ICONIP'2000, Taejon 2000, 565-569

[16] L. Kindermann, *MusiNum - The Music in the Numbers.* In: E.R. Miranda, Composing with Computers. Music Technology Series, Focal Press, Oxford 2001

[17] L. Kindermann, P. Protzel, *Computing iterative roots with second order training methods*. Proceedings of the International Joint Conference on Neural Networks (IJCNN'2001), Washington DC 2001

[18] L. Kindermann, A. Lewandowski, P. Protzel, *A framework for solving functional equations with neural networks.* Proceedings of the Eight International Conference on Neural Information Processing (ICONIP'2001), Fudan University Press, Shanghai 2001, 1075-1078

[19] L. Kindermann, P. Protzel, *Physics without laws - Making exact predictions with data based methods.* Proceedings of the International Joint Conference on Neural Networks (IJCNN'2002), Honolulu 2002, 1673-1677

[20] L. Kindermann, A. Lewandowski & P. Protzel, *Finding the Optimal Continuous Model for Discrete Data by Neural Network Interpolation of Fractional Iteration.* In: J.R. Dorronsoro (Ed.): Artificial Neural Networks - ICANN 2002 Proceedings. Lecture Notes in Computer Science, LNCS Vol 2415, Madrid 2002, 1094-1099

[21] L. Kindermann & Pando Georgiev, *Modelling Iterative Roots of Mappings in Multidimensional Spaces.* Proceedings of the Ninth International Conference on Neural Information Processing (ICONIP'2002), Singapore 2002

F.2 Patente

[22] L. Kindermann, P. Protzel, F. Schmid, O. Gramckow, V*erfahren und Einrichtung zur Bestimmung eines Zwischenprofils eines Metallbandes.* Deutsche Patentschrift DE 1998/0248D, 1998.

[23] L. Kindermann, P. Protzel, F. Schmid, O. Gramckow, SIEMENS AG, *Process and device for determining an intermediate section of a metal strip.* International Patent WO9942232, 1999.